

October 4, 2002

Grid Service Specification

Status of this Memo

This document provides information to the community regarding the specification of an OGSA Grid service. Distribution of this document is unlimited. This is a DRAFT document and continues to be revised.

Abstract

Building on both Grid and Web services technologies, the Open Grid Services Architecture (OGSA) defines mechanisms for creating, managing, and exchanging information among entities called *Grid services*. Succinctly, a Grid service is a Web service that conforms to a set of conventions (interfaces and behaviors) that define how a client interacts with a Grid service. These conventions, and other OGSA mechanisms associated with Grid service creation and discovery, provide for the controlled, fault resilient, and secure management of the distributed and often long-lived state that is commonly required in advanced distributed applications. In a separate document, we have presented in detail the motivation, requirements, structure, and applications that underlie OGSA. Here we focus on technical details, providing a full specification of the behaviors and Web Service Definition Language (WSDL) interfaces that define a Grid service.

**GLOBAL GRID FORUM****office@gridforum.org****www.ggf.org**

Full Copyright Notice

Copyright © Global Grid Forum (2002). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the GGF or other organizations, except as needed for the purpose of developing Grid Recommendations in which case the procedures for copyrights defined in the GGF Document process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the GGF or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE GLOBAL GRID FORUM DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property Statement

The GGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the GGF Secretariat.

The GGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the GGF Executive Director (see contact information at GGF website).

Contents

1	Introduction	5
2	Notational Conventions	5
3	Setting the Context	6
3.1	Relationship to Distributed Object Systems	6
3.2	Client-Side Programming Patterns	7
3.3	Relationship to Hosting Environment	8
4	The Grid Service	10
4.1	WSDL Extensions and Conventions	10
4.2	Service Description and Service Instance	11
4.3	Modeling Time in OGSA	12
4.4	Service Data Concept	12
4.4.1	serviceData	15
4.4.2	serviceDataDescription	17
4.4.3	serviceDataSet and Instance Service Data	19
4.4.4	XML Element Lifetime Declaration Properties	20
4.4.5	The wsdlLocation attribute	22
4.5	Interface Naming and Change Management	23
4.5.1	The Change Management Problem	23
4.5.2	Naming Conventions for Grid Service Descriptions	24
4.6	Naming Grid Service Instances: Handles and References	24
4.6.1	Grid Service Reference (GSR)	24
4.6.1.1	WSDL Encoding of a GSR	26
4.6.2	Grid Service Handle (GSH)	26
4.6.2.1	Service Instance Sameness	27
4.6.3	XML types for handles, references and locators	27
4.7	Grid Service Lifecycle	27
4.8	Common Handling of Operation Faults	28
5	Grid Service Interfaces	28
6	The GridService PortType	29
6.1	GridService PortType: Service Data Descriptions and Elements	29
6.2	GridService PortType: Operations and Messages	31
6.2.1	GridService :: FindServiceData	31
6.2.1.1	queryByServiceDataName	32
6.2.1.2	queryByXPath	32
6.2.1.3	queryByXQuery	32
6.2.2	GridService :: RequestTerminationAfter	32
6.2.3	GridService :: RequestTerminationBefore	33
6.2.4	GridService :: Destroy	34
7	The HandleResolver PortType	34
7.1	HandleResolver PortType: Service Data Descriptions	35
7.2	HandleResolver PortType: Operations and Messages	35
7.2.1	HandleResolver :: FindByHandle	35
8	Notification	36
8.1	The NotificationSource PortType	37
8.1.1	NotificationSource PortType: Service Data Descriptions and Elements ..	37

8.1.2	NotificationSource PortType: Operations and Messages	37
8.1.2.1	NotificationSource :: Subscribe	37
8.2	The NotificationSubscription PortType	39
8.2.1	NotificationSubscription PortType: Service Data Descriptions	39
8.2.2	NotificationSubscription PortType: Operations and Messages	39
8.3	The NotificationSink PortType	39
8.3.1	NotificationSink PortType: Service Data Descriptions	40
8.3.2	NotificationSink PortType: Operations and Messages	40
8.3.2.1	NotificationSink :: DeliverNotification.....	40
8.4	Integration With Notification Intermediaries.....	40
9	The Factory PortType.....	41
9.1	Factory PortType: Service Data Descriptions.....	42
9.2	Factory PortType: Operations and Messages.....	42
9.2.1	Factory :: CreateService.....	42
10	Registration	43
10.1	WS-Inspection Document	43
10.2	The Registration portType	43
10.2.1	Registration PortType: Service Data Descriptions.....	44
10.2.2	Registration PortType: Operations and Messages	44
10.2.2.1	Registration :: RegisterService.....	44
10.2.2.2	Registration :: UnregisterService	45
11	Security Considerations.....	45
12	Change Log	45
12.1	Draft 1 (2/15/2002) → Draft 2 (6/13/2002)	45
12.2	Draft 2 (6/13/2002) → Draft 3 (07/17/2002)	46
12.3	Draft 3 (07/17/2002) → Draft 4 (10/1/2002)	46
13	Author Information	47
14	Acknowledgements	48
15	Glossary.....	48
16	References	48
16.1	Normative References	48
16.2	Informative References	49
17	XML and WSDL Specifications	50

1 Introduction

The *Open Grid Services Architecture* (OGSA) [Grid Physiology] integrates key Grid technologies [Grid Book, Grid Anatomy] (including the Globus Toolkit [Globus Overview]) with Web services mechanisms [Web Services Book] to create a distributed system framework based around the *Grid service*. A *Grid service instance* is a (potentially transient) service that conforms to a set of conventions (expressed as WSDL interfaces, extensions, and behaviors) for such purposes as lifetime management, discovery of characteristics, notification, and so forth. Grid services provide for the controlled management of the distributed and often long-lived state that is commonly required in sophisticated distributed applications. OGSA also introduces standard factory and registration interfaces for creating and discovering Grid services.

In this document, we propose detailed specifications for the conventions that govern how clients create, discover, and interact with a Grid service. That is, we specify (a) how Grid service instances are named and referenced, (b) the interfaces (and associated behaviors) that define any Grid service and (c) the additional (optional) interfaces and behaviors associated with factories and registries. We do *not* address how Grid services are created, managed, and destroyed within any particular hosting environment. Thus, services that conform to this specification are not necessarily portable to various hosting environments, but they can be invoked by any client that conforms to this specification (of course, subject to policy and compatible protocol bindings).

Our presentation here is deliberately terse, in order to avoid overlap with [Grid Physiology]. The reader is referred to [Grid Physiology] for discussion of motivation, requirements, architecture, relationship to Grid and Web services technologies, other related work, and applications.

This document is a work in progress and feedback is encouraged. Future versions will incorporate additional pedagogical text and examples. We also draw the reader's attention to various "Notes" that indicates areas of particular uncertainty.

2 Notational Conventions

The key words "MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," and "OPTIONAL" are to be interpreted as described in RFC-2119 [RFC 2119].

This specification uses namespace prefixes throughout; they are listed in Table 1. Note that the choice of any namespace prefix is arbitrary and not semantically significant.

Table 1: Prefixes and Namespaces used in this specification.

Prefix	Namespace
gsdl	"http://www.gridforum.org/namespaces/2002/07/gridServices"
wsdl	"http://www.w3.org/2002/07/wsdl"
wsil	"http://schemas.xmlsoap.org/ws/2001/10/inspection/"
http	"http://www.w3.org/2002/06/wsdl/http"
xsd	"http://www.w3.org/2001/XMLSchema"
xsi	"http://www.w3.org/2001/XMLSchema-instance"

Namespace names of the general form "http://example.org/..." and "http://example.com/..." represent application or context-dependent URIs [RFC 2396].

In this document we use **bold** font face to emphasize WSDL defined elements and properties, as well as elements and properties of WSDL extensions defined in this document.

The following abbreviations and terms are used in this document:

- *GSH*: Grid Service Handle, as defined in Section 4.6.
- *GSR*: Grid Service Reference, as defined in Section 4.6.
- *SDE*: Service Data Element, as defined in Section 4.3.
- *SDD*: Service Data Description, as defined in Section 4.3.
- The terms *Web services*, *XML*, *SOAP*, *WSDL*, and *WS-Inspection* are as defined in [Grid Physiology].

The term *hosting environment* is used in this document to denote the server in which one or more Grid service implementations run. Such servers are typically language and/or platform specific. Examples include native Unix and Windows processes, J2EE application servers, and Microsoft .NET.

Unresolved issues with the specification are interspersed in appropriate locations through this specification, are highlighted in yellow, and begin with "Issue N:", where N is the GGF OGSi working group bugzilla database bug number for this issue. This database is located at <http://www.gridforum.org/ogsi-wg/bugzilla>.

3 Setting the Context

Although [Grid Physiology] describes overall motivation for the Open Grid Services Architecture, this document describes the architecture at a more detailed level. Correspondingly, there are several details we examine in this section that help put the remainder of the document in context. Specifically, we discuss the relationship between OGSA and distributed object systems, and also the relationship that we expect to exist between OGSA and the existing Web services framework, examining both the client-side programming patterns and a conceptual hosting environment for Grid services.

We emphasize that the patterns described in this section are enabled but not *required* by OGSA. We discuss these patterns in this section to help put into context certain details described in the other parts of this document.

3.1 Relationship to Distributed Object Systems

As we describe in much more detail below, a given Grid service implementation is an addressable, and potentially stateful, instance that implements one or more interfaces described by WSDL **portTypes**. Grid service factories (Section 9) can be used to create instances of a given **portType**. Each Grid service instance has a unique identity with respect to the other instances in the system. Each instance can be characterized as state coupled with behavior published through type-specific operations. The architecture also supports introspection in that a client application can ask a Grid service instance to return information describing itself, such as the collection of **portTypes** that it implements.

Grid service instances are made accessible to (potentially remote) client applications through the use of a Grid Service Handle (Section 4.6.2) and a Grid Service Reference (Section 4.6.1). These constructs are basically network-wide pointers to specific Grid service instances hosted in

(potentially remote) execution environments. A client application can use a Grid Service Reference to send requests (represented by the operations defined in the **portType**(s) of the target service) directly to the specific instance at the specified network-attached service endpoint identified by the Grid Service Reference.

Each of the characteristics introduced above (stateful instances, typed interfaces, unique global names, etc.) is frequently also cited as a fundamental characteristic of so-called *distributed object-based systems*. However, there are also various other aspects of distributed object models (as traditionally defined) that are specifically *not* required or prescribed by OGSA. For this reason, we do not adopt the term distributed object model or distributed object system when describing this work, but instead use the term Open Grid Services Architecture, thus emphasizing the connections that we establish with both Web services and Grid technologies.

Among the object-related issues that are not addressed within OGSA are implementation inheritance, service mobility, development approach, and hosting technology. The Grid service specification does not require, nor does it prevent, implementations based upon object technologies that support inheritance at either the interface or the implementation level. There is no requirement in the architecture to expose the notion of implementation inheritance either at the client side or the service provider side of the usage contract. In addition, the Grid service specification does not prescribe, dictate, or prevent the use of any particular development approach or hosting technology for the Grid service. Grid service providers are free to implement the semantic contract of the service in any technology and hosting architecture of their choosing. We envision implementations in J2EE, .NET, traditional commercial transaction management servers, traditional procedural UNIX servers, etc. We also envision service implementations in a wide variety of programming languages that would include both object-oriented and non-object-oriented alternatives.

3.2 Client-Side Programming Patterns

Another important issue that we feel requires some explanation, particularly for readers not familiar with Web services, is how OGSA interfaces are likely to be invoked from client applications. OGSA incorporates an important component of the Web services framework: the use of WSDL to describe multiple protocol bindings, encoding styles, messaging styles (RPC vs. document-oriented), and so on, for a given Web service.

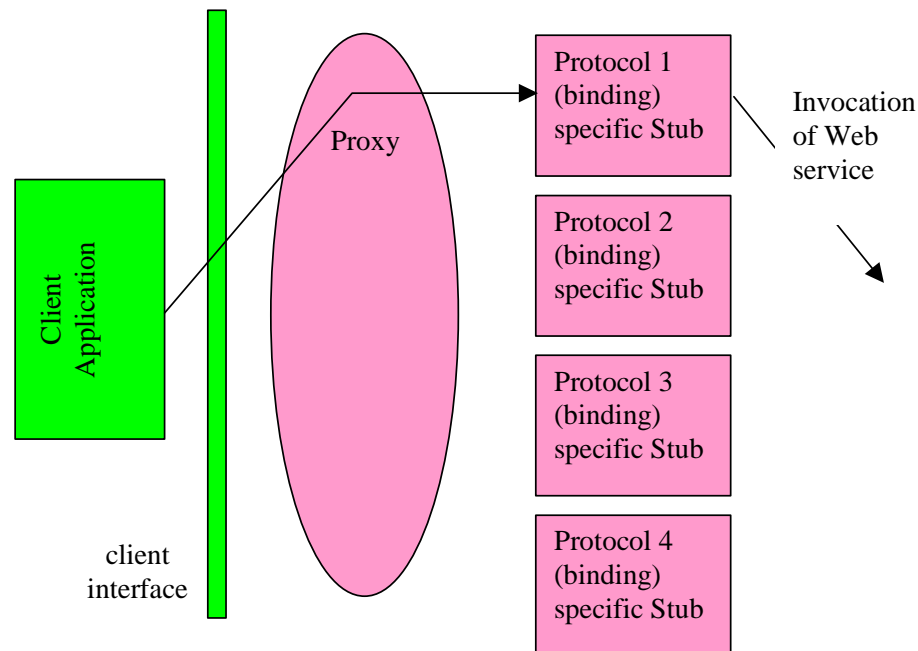


Figure 1: A possible client-side runtime architecture

Figure 1 depicts a possible (but not required) client-side architecture for OGSA. In this approach, there is a clear separation between the client application and the client-side representation of the Web service (proxy), including components for marshalling the invocation of a Web service over a chosen binding. In particular, the client application is insulated from the details of the Web service invocation by a higher-level abstraction: the client-side interface. Various runtime tools can take the WSDL description of the Web service and generate interface definitions in a wide-range of programming language specific constructs (e.g. Java interfaces). This interface is a front-end to specific parameter marshalling and message routing that can incorporate various binding options provided by the WSDL. Further, this approach allows certain efficiencies, for example, detecting that the client and the Web service exist on the same network host, and therefore avoiding the overhead of preparing for and executing the invocation using network protocols. One example of this approach to Web services is Java API for XML-Based RPC [JAX-RPC].

Within the client application runtime, a *proxy* provides a client-side representation of remote service instance's interface. Proxy behaviors specific to a particular encoding and network protocol (*binding* in Web services terminology) are encapsulated in a *protocol (binding)-specific stub*. Details related to the binding-specific access to the Grid service, such as correct formatting and authentication mechanics, happen here; thus, the application is not required to handle these details itself.

We note that it is possible, but not recommended, for developers to build customized code that directly couples client applications to fixed bindings of a particular Grid service. Although certain circumstances demand potential efficiencies gained this style of customization, this approach introduces significant inflexibility into the system and therefore should be used under extraordinary circumstances.

3.3 Relationship to Hosting Environment

OGSA does not dictate a particular service provider-side implementation architecture. A variety of approaches are possible, ranging from implementing the Grid service directly as an operating system process to a sophisticated server-side component model such as J2EE. In the former case,

most or even all support for standard Grid service behaviors (invocation, lifetime management, registration, etc.) is encapsulated within the user process, for example via linking with a standard library; in the latter case, many of these behaviors will be supported by the hosting environment.

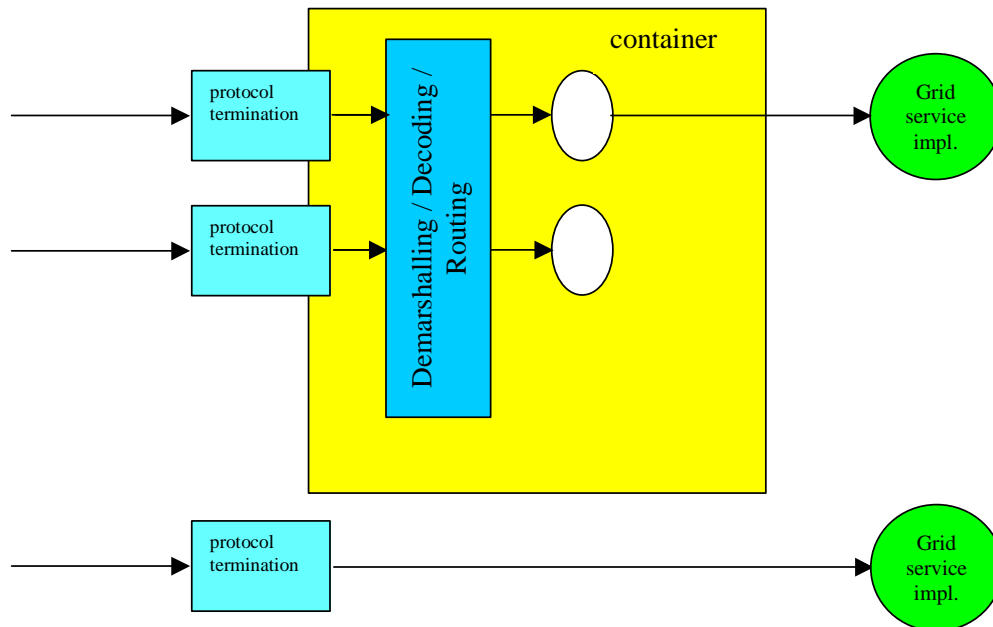


Figure 2: Two alternative approaches to the implementation of argument demarshalling functions in a Grid Service hosting environment

Figure 2 illustrates these differences by showing two different approaches to the implementation of argument demarshalling functions. We assume that, as is the case for many Grid services, the invocation message is received at a network protocol termination point (e.g., an HTTP servlet engine), which converts the data in the invocation message into a format consumable by the hosting environment. At the top of Figure 2, we illustrate two Grid services (the ovals) associated with container-managed components (for example EJBs within a J2EE container). Here, the message is dispatched to these components, with the container frequently providing facilities for demarshalling and decoding the incoming message from a format (such as an XML/SOAP message) into an invocation of the component in native programming language. In some circumstances (the lower oval), the entire behavior of a Grid service is completely encapsulated within the component. In other cases (the upper oval), a component will collaborate with other server-side executables, perhaps through an adapter layer, to complete the implementation of the Grid services behavior. At the bottom of Figure 2, we depict another scenario wherein the entire behavior of the Grid service, including the demarshalling/decoding of the network message, has been encapsulated within a single executable. Although this approach may have some efficiency advantages, it provides little opportunity for reuse of functionality between Grid service implementations.

A container implementation may provide a range of functionality beyond simple argument demarshalling. For example, the container implementation may provide lifetime management functions, intercepting lifetime management functions and terminating service instances when a service lifetime expires or an explicit destruction request is received. Thus, we avoid the need to re-implement these common behaviors in different Grid service implementations.

4 The Grid Service

The purpose of this document is to specify the interfaces and behaviors that define a *Grid service*. In brief, a *Grid service* is a WSDL-defined service that conforms to a set of conventions relating to its interface definitions and behaviors. In this section, we expand upon this brief statement by:

- Introducing a set of WSDL conventions that we make use of in our Grid service specification;
- Defining *Grid service description* and *Grid service instance*, as organizing principles for the extensions and their use;
- Defining how time is modeled in OGSA;
- Defining *service data*, which provides a standard way for representing and querying meta-data and state data from a service instance;
- Defining the Grid Service Handle and Grid Service Reference constructs, which we use to refer to Grid service instances;
- Providing example WSDL documents illustrating the extensibility elements;
- Defining a common approach for conveying fault information from operations;
- Defining the lifecycle of a Grid service instance.

In subsequent sections, we introduce various **portTypes**, starting with the GridService **portType** that must be supported by any Grid service, and then proceeding to HandleResolver, Notification, and the remainder of the portTypes that describe fundamental behavior of Grid services.

4.1 WSDL Extensions and Conventions

Web services technologies are designed to support loosely coupled, coarse-grained dynamic systems. As such, they do not fully address all needs of the types of distributed systems that OGSA is defined to support. To close this gap, this specification defines a set of WSDL extensions (defined using extensibility elements allowed by the WSDL language) and conventions on the use of Web services, which we list in Table 2 and define in more detail in subsequent sections. We emphasize that the extensions are not specific to Grid computing per se, but have general applicability within Web services as a means of structuring complex and long-lived stateful applications. We advocate their adoption within the broader Web services standards bodies such as the W3C.

Table 2: Proposed WSDL conventions and extensions introduced by OGSA

Concept	WSDL Element	Brief Description	See Section
Service state data and meta-data	serviceData	serviceData elements represent properties of the service's state that may be externally queried. Some serviceData elements may represent	4.3

		serviceData elements may appear as extensions of the service's description, as part of the service's type.	
Service data description	serviceData Description	Formal descriptions of serviceData elements. These descriptions appear as extensions of the Grid service's WSDL.	4.4.2
Interface Naming	convention on portType name	Naming conventions and immutability of portType names.	4.2
Grid Service Reference	N/A	Mechanism to convey capabilities of a service to a client. <i>Can</i> be a WSDL document.	4.6.1
Grid Service Handle	N/A	Conventional use of URI to act as unique identifier of a Grid service instance.	4.6.2

This draft is based on extensions to the WSDL language proposed by the W3C Web Services Description Working Group [WSDL 1.2]. In particular, we are relying upon the following new constructs proposed for WSDL 1.2 (draft):

- open content model (extensibility elements appearing in each WSDL element)
- **portType** extension.

4.2 Service Description and Service Instance

We distinguish in OGSA between the *description* of a Grid service and an *instance* of a Grid service:

- A *Grid service description* describes how a client interacts with service instances. This description is independent of any particular instance. Within a WSDL document, the Grid service description is embodied in the most derived portType (i.e. the portType referenced by the wsdl:service element describing the service) of the instance, along with its associated **portTypes**, **serviceDataDescriptions**, **messages**, and **types** definitions.
- A Grid service description may be simultaneously used by any number of *Grid service instances*, each of which:
 - embodies some state with which the service description describes how to interact;
 - has one or more unique Grid Service Handles;
 - and has one or more Grid Service References to it.

A common form of Grid Service Reference (defined in section 4.6.1) is a WSDL document comprising a **service** element, which carries an attribute that refers to a most derived portType defined by the service description of that instance.

A service description is primarily used for two purposes. First, as a description of a service interface, it can be used by tooling to automatically generate client interface proxies, server skeletons, etc. Second, it can be used for discovery, for example, to find a service instance that

implements a particular service description, or to find a factory that can create instances with a particular service description.

The service description is meant to capture both interface syntax, as well as (in a very rudimentary, non-normative fashion) semantics. Interface syntax is, of course, described by **portTypes**.

Semantics may be inferred through the name assigned to the **portType**. For example, when defining a Grid service, one defines zero or more uniquely named **portTypes**, and then collects a set of **portTypes** defined from a variety of sources into a final or most derived portType. Concise semantics can be associated with each of these names in specification documents – and perhaps in the future through Semantic Web or other formal descriptions. These names can then be used by clients to discover services with the sought-after semantics, by searching for service instances and factories with the appropriate names. Of course, the use of namespaces to define these names provides a vehicle for assuring globally unique names.

4.3 Modeling Time in OGSA

Issue 55: WS-Timestamp is a component of WS-Security that is being standardized in OASIS. We need to examine how OGSi is modelling time and reconcile.

Throughout this specification there is the need to represent time that is meaningful to multiple parties in the distributed Grid. For example: information may be tagged by a producer with timestamps in order to convey that information's useful lifetime to consumers; clients need to negotiate service instance and registration lifetimes with services; and multiple services may need a common understanding of time in order for clients to be able to manage their simultaneous use and interaction.

The GMT global time standard is assumed for Grid services, allowing operations to refer unambiguously to absolute times. However, assuming the GMT time standard to represent time does *not* imply any particular level of clock synchronization between clients and services in the Grid. In fact, no specific accuracy of synchronization is specified or expected by this specification, as this is a service-quality issue.

Grid service hosting environments and clients SHOULD utilize the Network Time Protocol (NTP) or equivalent function to synchronize their clocks to the global standard GMT time. However, clients and services MUST accept and act appropriately on messages containing time values that might be out of range due to inadequate synchronization, where "appropriately" MAY include refusing the use the information associated with those time values. Furthermore, clients and services requiring global ordering or synchronization at a finer granularity than their clock accuracies or resolutions allow for MUST coordinate through the use of additional synchronization service interfaces, such as through transactions or synthesized global clocks.

4.4 Service Data Concept

Issue 56: There continues to be confusion over the use of and motivation for the serviceData concept. We need to add non-normative text to clarify why an existing approach with operations getXXX, where XXX is the name of a variable, is not sufficient.

In order to support discovery, introspection, and monitoring of Grid service instances, we introduce the concept of *service data*, which refers to descriptive information about a Grid service instance, including both *meta-data* (information about the service instance) and *state data* (runtime properties of the service instance). We begin with a non-normative review of the components of the service data concept and their relationships.

Each Grid service instance has an associated set of *service data elements (SDEs)*. Each SDE is represented in XML by a **serviceData** element (see Section 4.4.1). Each SDE can be used to model a separate property of the service:

```
<serviceData name="gsdl:portType"> ...
<serviceData name="gsdl:queryExpressionTypes"> ...
<serviceData name="tns:configuration"> ...
<serviceData name="tns:cpuSpeed"> ...
...
```

ServiceData elements are roughly analogous to instance variables in a class definition in some object-oriented programming language. However, service data elements represent read-only state data.

Issue 51: Writeable SDEs. Should we add an operation to Grid service to allow clients to change serviceData?

A **serviceData** element is a named container that **MUST** contain zero or more XML elements of some XML type. We refer to each XML element as a *service data value element*, and the complete set of elements within an SDE as the *service data value*. The service data value in the following serviceData element consists of 3 service data value elements:

```
<serviceData name="gsdl:queryExpressionTypes">
  <queryExpressionType>gsdl:queryByServiceDataName
</queryExpressionType>
  <queryExpressionType>gsdl:queryByXQuery</queryExpressionType>
  <queryExpressionType>sql:queryBySQL</queryExpressionType>
</serviceData>
```

Note: all of the service data value elements in the service data element are of the same XML type (in this case, gsdl:queryExpressionType).

Each Grid service instance is associated with a set of SDEs. This set is modeled as a **serviceDataSet** (see Section 4.4.3). The following is an example of a **serviceDataSet** for some Grid service instance:

```
<serviceDataSet>
  <serviceData name="tns:cpuSpeed">
    <gsdl:float>12.5</gsdl:float>
  </serviceData>
  <serviceData name="gsdl:queryExpressionTypes">
    <queryExpressionType>gsdl:queryByServiceDataName
    </queryExpressionType>
    <queryExpressionType>gsdl:queryByXQuery</queryExpressionType>
    <queryExpressionType>sql:queryBySQL</queryExpressionType>
  </serviceData>
  <serviceData name="tns:configuration">
    <tns:cpuConfig> ...
    </tns:cpuConfig>
  </serviceData>
  ...
</serviceDataSet>
```

There are two ways that SDEs are associated with a Grid service instance's **serviceDataSet**: SDEs that appear in the Grid service's description and SDEs that are dynamically added to the Grid service at runtime.

Those SDEs that appear in a Grid service's service description are called **structural SDEs**. Structural SDEs are **serviceData** elements that appear as extensibility elements in the **portType** elements associated with a service description.

```
<wsdl:portType name="cpuResource">
  <serviceData name="tns:manufacturer" ...
  <serviceData name="tns:cpuSpeed" ...
  <serviceData name="tns:configuration" ...
...
</wsdl:portType>
```

A structural SDE declares that any Grid service instance implementing the given **portType** MUST include an SDE of the same name in its **serviceDataSet**. The **serviceData** element that appears in a service description MAY also indicate the initial service data value for the SDE.

```
<wsdl:portType name="CPU">
...
  <serviceData name="tns:manufacturer" ...>
    <tns:manufName>chipsRUs.com</tns:manufName>
  </serviceData>
...
```

The SDE value of many SDEs MAY change over the lifetime of the Grid service instance. For some SDEs, its value never changes at runtime; others may change SDE value frequently.

SDEs that are added to the Grid service instance's **serviceDataSet** at runtime are called *non-structural* SDEs. Non-structural SDEs are *not* declared within the Grid service instance's service description. Otherwise, non-structural SDEs are the same as structural SDEs. Thus a Grid service instance's **serviceDataSet** contains the complete set of SDEs (both structural and non-structural) associated with that Grid service instance. A client can determine the complete set of SDEs associated with any Grid service instance by examining the ServiceDataNames SDE defined in the GridService portType (See Section 6.1).

A Grid service's **serviceDataSet** MAY be accessible to clients in two ways. A Grid service instance MUST implement the **FindServiceData** operation of the GridService **portType** (see Section 6.2.1), which provides a simple, extensible, client-initiated query against the instance's **serviceDataSet**. A Grid service instance MAY additionally implement the **NotificationSource portType** that provides the **Subscribe** operation (see Section 8.1.2.1). This operation enables a client to ask a Grid service instance to notify it of subsequent changes to its **serviceDataSet**.

The service data that is available for query or subscription by a client MAY be subject to policy restrictions. This implies, for example:

- Some service data elements MAY not be available to some clients.
- Some service data value elements within an SDE MAY not be available to some clients. Therefore, a client MAY perceive the service data element as having fewer value elements than specified by minOccurs (see 4.4.2).
- Two different clients MAY see different service data value elements in the same **serviceData** element from the same service.

The characteristics of each service data element **MUST** be declared using a **serviceDataDescription** element (see Section 4.4.2). Such a declaration, called a *service data description (SDD)*, specifies properties such as the name of the SDE, the XML type of the service data value elements, how many times service data value elements may occur, whether the value elements may change during the lifetime of the instance, etc. **serviceDataDescription** elements **MAY** appear as part of a Grid service's service description, as an extension of the **definitions** element. For example, the description:

```
<wsdl:definitions ...>
  <serviceDataDescription name="cpuLoad" element="gsdl:float" ...>
...
</wsdl:definitions>
```

Defines a template for an SDE container that would contain XML elements of type `gsdl:float`.

Each SDD has a name that **MUST** be unique amongst all **serviceDataDescription** elements within its namespace. This name, when prepended with the URI of the enclosing namespace forms a qname that is globally unique. The name property of a **serviceData** element **MUST** correspond to the name of the SDD to which that SDE conforms.

4.4.1 serviceData

A **serviceData** element **MAY** appear as part of a **portType** and/or as part of a service instance's **serviceDataSet** (see Section 4.4.3). A **serviceData** element is a container for a collection of service data value elements.

A **serviceData** element has the following non-normative grammar:

```
<gsdl:serviceData
  name="qname"
  <-- extensibility attribute -->* >
  <-- extensibility element -->*
</gsdl:serviceData>
```

Each **serviceData** element contains the following information:

- **name:** This attribute's value is the qname of the **serviceDataDescription** (see Section 4.4.2) element that describes (defines the template for) this **serviceData** element. The **serviceDataDescription** may come from any namespace.
- *Extensibility attributes:* A **serviceData** element **MAY** have other extensibility attributes, including but not limited to:
 - *Lifetime declarations:* As defined in Section 4.4.4, **gsdl:goodFrom**, **gsdl:goodUntil**, and **gsdl:availableUntil** attributes **MAY** be placed on a **serviceData** element to declare the lifetime characteristics of that service data element and its value.
 - *Application-specific:* A **serviceData** element **MAY** have additional, application-specific extensibility attributes from any namespace.
- *Extensibility elements:* A **serviceData** element **MAY** have other extensibility elements including but not limited to:
 - *Service data value elements:* The **serviceData** extensibility element **MUST** include zero or more elements that conform to the XML element definition referred to by the

element property in the **serviceDataDescription** referred to by the **name** property. The number of such service data value elements **MUST** be greater than or equal to the value specified by the **minOccurs** property of the corresponding **serviceDataDescription** element, and **MUST** be less than or equal to the value specified by the **maxOccurs** property of that same **serviceDataDescription** element. These **serviceDataDescription** properties govern the number of service data value elements that appear in a **serviceData** element within a service instance's **serviceDataSet**, and do not apply to a **serviceData** element that appears in **portType** elements.

- *Application-specific:* A **serviceData** element **MAY** have additional extensibility elements from any namespace.

For structural SDEs, the initial value of the SDE **MAY** be determined by SDE values declared in one or more **portTypes** associated with the Grid service description.

The following is an example of four **serviceData** declarations in a **portType**:

```
<portType name="CPU" extends "gsdl:GridService">
  ...
  <gsdl:serviceData name="tns:CPUSpeed"/>

  <gsdl:serviceData name="tns:CPULoad">
    <gsdl:float>0.00</gsdl:float>
  </gsdl:serviceData>

  <gsdl:serviceData name="tns:manufacturer" ...>
    <tns:manufName>chipsRUs.com</tns:manufName>
  </gsdl:serviceData>

  <gsdl:serviceData name="gsdl:queryExpressionTypes">
    <gsdl:queryExpressionType>
      gsdl:queryByXQuery
    </gsdl:queryExpressionType>
    <gsdl:queryExpressionType>
      sql:queryBySQL
    </gsdl:queryExpressionType>
  </gsdl:serviceData>

  ...
</portType>
```

The SDDs that correspond to these **serviceData** elements appears in Section 4.4.2. This first **serviceData** element declares that any instance that implements the “CPU” **portType** **MUST** have a **serviceData** element named “tns:CPUSpeed” in its **serviceDataSet**. The second **serviceData** element, in addition to declaring that the “tns:CPULoad” **serviceData** element **MUST** appear in instance, also defines an initial service data value element for this **serviceData** element. This value element **MAY** be used as part of the service description, for example during discovery. The **serviceDataDescription**'s **mutability** attribute (See Section 4.4.2) determines what happens to this value element in instances that implement this **portType**.

The third SDE declaration, “tns:manufacturerName” is similar in spirit to “tns:cpuSpeed”.

The fourth SDE declaration, “gsdl:queryExpressionTypes” is a good example of the impact of **portType** extension on service data. The CPU **portType** is declared to be an extension of

GridService portType. As shown in Section 6.1, the GridService portType itself declares an SDE of name “gsdl:queryExpressionTypes”:

```
<gsdl:serviceData name="gsdl:QueryExpressionTypes">
  <gsdl:queryExpressionType>
    gsdl:queryByServiceDataName
  </gsdl:queryExpressionType>
</gsdl:serviceData>
```

Note that in this case, both the CPU portType and the GridService portType declares an SDE named gsdl:QueryExpressionTypes. When multiple portTypes within a Grid service’s description declare the same SDE, the SDE appears only once within any instance’s **serviceDataSet**. Therefore the initial set of SDEs is the union of all the SDEs declared in the all the portTypes within a Grid service’s service description. Of course, the Grid service instance’s **serviceDataSet** MAY be extended with additional (non-structural) SDEs.

Further, both portTypes declare initial value for the gsdl:QueryExpressionTypes SDE. In these cases, the initial value for the SDE in any instance’s **serviceDataSet** is the concatenation of all the SDE value elements in each SDE declaration. The ordering of the concatenation is not defined, implementations are free to use any ordering. In our example, the initial value of the SDE for any Grid service instance implementing the CPU portType would be:

```
<gsdl:serviceData name="gsdl:QueryExpressionTypes">
  <gsdl:queryExpressionType>
    gsdl:queryByServiceDataName
  </gsdl:queryExpressionType>
  <gsdl:queryExpressionType>
    gsdl:queryByXQuery
  </gsdl:queryExpressionType>
  <gsdl:queryExpressionType>
    sql:queryBySQL
  </gsdl:queryExpressionType>
</gsdl:serviceData>
```

The set of SDE values that appear in the service description determine the initial SDE value. The SDE value MAY change during the lifetime of the Grid service instance.

4.4.2 serviceDataDescription

The **serviceDataDescription** element MUST appear as an extension of the WSDL **definitions** element. It is used to describe service data elements.

The **serviceDataDescription** element has the following non-normative grammar:

```
<gsdl:serviceDataDescription
  name="NCName"
  element="qname"
  minOccurs="nonNegativeInteger"?
  maxOccurs=("nonNegativeInteger" | "unbounded")?
  instanceOnly="boolean"?
  mutability="constant" | "append" | "mutable"?>
  <wsdl:documentation .... />?
  <-- extensibility element --> *
</gsdl:serviceDataDescription>
```

Each **serviceDataDescription** element contains the following information:

- **name**: A name for this service data description, which **MUST** be unique amongst all **serviceDataDescription** names within the namespace in which it is defined.
- **element**: The qualified name of the XML element definition whose instances **MAY** appear as service data value elements contained in any **serviceData** element that conforms to this **serviceDataDescription**. Note: a **wsdl:import** element **MAY** be used to suggest a possible location for further information about the namespaces associated with the value of the **element** property.
- **minOccurs**: The minimum number of service data value elements, each conforming to the XML element defined by the **element** attribute, which **MUST** be contained in an SDE that conforms to this SDD. If this attribute is omitted, then it defaults to 1.
- **maxOccurs**: The maximum number of service data value elements, each conforming to the XML element defined by the **element** attribute, which **MUST** be contained in an SDE that conforms to this SDD. If this attribute is omitted, then it defaults to 1.
- **instanceOnly**: Structural SDEs should not be assigned an initial value in the service description; the initial SDE value **MUST** be assigned after the Grid service instance is created.
- **mutability**: The value of this attribute determines the way in which developers can expect the SDE value to change during runtime. For structural SDEs, the initial SDE value is determined by the service description. For non-structural SDEs the initial value is determined when the SDE is associated with the Grid service instance's **serviceDataSet**. There are three possible values for this attribute:
 - **mutability="constant"**: this implies that the SDE value should not change during the lifetime of the Grid service instance. For some SDEs, the initial SDE value is non-empty. In this case, the initial SDE value is exactly the SDE value for the entire lifetime of the instance. In the case where the initial SDE value is empty, it is permissible, at initialization time, for the instance to assign an initial SDE value. After this initial assignment, the set **MUST NOT** change for the lifetime of the Grid service instance.
 - **mutability="append"**: this implies the elements in the initial SDE value is guaranteed to be part of the SDE value for the lifetime of the Grid service. Additional elements can be added to the SDE value, but once these elements are added, they cannot be removed.
 Issue: 46 There is a possible additional mutability enum value, that might deal with "loose append" ie once something is added, it may subsequently disappear from the set.
 - **mutability="mutable"**: this implies any of the element in the SDE value may be removed at anytime, and others may be added. There is no statement of guarantee that any of the initial value set is present in the SDE.

Some example SDDs:

```
<gsdl:serviceDataDescription
  name="CPUSpeed"
  element="gsdl:float"
  minOccurs="1"
  maxOccurs="1"
  mutability="mutable">
  <wsdl:documentation>
```

```

    Example definition of a measurement of CPU speed
    </wsdl:documentation>
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription
  name="CPULoad"
  element="gsdl:float"
  minOccurs="1"
  maxOccurs="1"
  mutability="mutable">
  <wsdl:documentation>
    Example definition of a measurement of CPU load
  </wsdl:documentation>
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription>
  name="manufacturer"
  element="tns:manufName"
  minOccurs="1"
  maxOccurs="1"
  mutability="constant">
  <wsdl:documentation>
    The name of the manufacturer. This is fixed at initialization
    time and does not change.
  </wsdl:documentation>
</gsdl:serviceDataDescription>

```

4.4.3 serviceDataSet and Instance Service Data

A set of **serviceData** elements MAY be aggregated into a **serviceDataSet** element, which has the following non-normative grammar:

```

<gsdl:serviceDataSet>
  <gsdl:serviceData ...> *
  ...
</gsdl:serviceDataSet>

```

Each Grid service instance **MUST** make available exactly one **serviceDataSet** element, against which it evaluates all **FindServiceData** (Section 6.2.1) and **Subscribe** (Section 8.1.2.1) operation requests from its clients.

An instance's **serviceDataSet** element **MUST** include all **serviceData** elements declared in the instance's **portType(s)**. The **serviceDataSet** MAY also include additional **serviceData** elements that are not declared in the instance's **portType** (i.e. non-structural SDEs).

This specification does *not* dictate how the service data set and its elements are represented internally within the runtime of a Grid service instance. The GridService **portType** (Section 6) provides a **FindServiceData** operation that allows clients to issue queries against this collection of *logical* XML elements, and the **NotificationSource portType** (Section 8.1) provides a **Subscribe** operation that allows clients to subscribe to changes to this same *logical* collection. We use the term *logical* since there is no requirement for the Grid service implementation to actually maintain the service data set and its elements in a persistent form. Instead, a Grid service instance MAY choose to create the XML elements dynamically from other data sources at the time the **FindServiceData** operation is invoked, and as necessary to generate notification messages as a result of the **Subscribe** operation.

4.4.4 XML Element Lifetime Declaration Properties

Since service data elements may represent point-in-time observations of dynamic state of a service instance, it is critical that consumers of service data be able to understand the valid lifetimes of these observations. The client MAY use this time-related information to reason about the validity and availability of the **serviceData** element and its value, though the client is free to ignore the information at its own discretion.

We define three XML attributes, which together describe the lifetimes associated with an XML element and its sub-elements. These attributes MAY be used in any XML element that allows for extensibility attributes, including the **serviceData** element.

The three lifetime declaration properties are:

- **gsdl:goodFrom="xsd:dateTime"**: Declares the time from which the content of the element is said to be valid. This is typically the time at which the value was created.
- **gsdl:goodUntil="xsd:dateTime"**: Declares the time until which the content of the element is said to be valid. This property MUST be greater than or equal to the **goodFrom** time.
- **gsdl:availableUntil="xsd:dateTime"**: Declares the time until which this element itself is expected to be available, perhaps with updated values. Prior to this time, a client SHOULD be able to obtain an updated copy of this element. After this time, a client MAY no longer be able to get a copy of this element. This property MUST be greater than or equal to the **goodFrom** time.

We use the following **serviceData** element example to illustrate and further define these lifetime declaration attributes:

```
<wsdl:definitions
  targetNamespace="http://example.com/ns"
  xmlns:n1="http://example.com/ns"
  ... >

  <wsdl:types>
    <xsd:schema ...
      targetNamespace=http://example.com/ns
      ...>
      <xsd:element name="e1" type="n1:MyType"/>
      <xsd:complexType name="MyType">
        <xsd:sequence>
          <xsd:element name="e2" type="xsd:string" minOccurs="1"/>
          <xsd:element name="e3" type="xsd:string" minOccurs="1"/>
          <xsd:element name="e4" type="xsd:string" minOccurs="0"/>
        </xsd:sequence>
        <anyAttribute namespace="##any"/>
      </xsd:complexType>
    </xsd:schema>
  </wsdl:types>
  ...
  <gsdl:serviceDataDescription
    name="MySDE"
    element="n1:e1"
    minOccurs="1"
    maxOccurs="1"
    mutability="mutable"/>
  ...
```

```

<wsdl:portType name="MyPortType">
  ...
  <gsdl:serviceData name="n1:MySDE" />
  ...
</wsdl:portType>
...
</wsdl:definitions>

```

And within the service instance's **serviceDataSet**:

```

<gsdl:serviceData
  name="n1:MySDE"
  goodFrom="2002-04-27T10:20:00.000-06:00"
  goodUntil="2002-04-27T11:20:00.000-06:00"
  availableUntil="2002-04-28T10:20:00.000-06:00">
  <n1:e1>
    <n1:e2>
      abc
    </n1:e2>
    <n1:e3 gsdl:goodUntil="2002-04-27T10:30:00.000-06:00">
      def
    </n1:e3>
    <n1:e4 gsdl:availableUntil="2002-04-27T20:20:00.000-06:00">
      ghi
    </n1:e4>
  </n1:e1>
</gsdl:serviceData>

```

The **goodFrom** and **goodUntil** attributes of the **serviceData** element refer to the service data value contained in the **serviceData** element's extensibility element, which in this example is the element **n1:e1**, which conforms to the type "n1:MyType". These attributes declare to the consumer of this SDE what the expected lifetime is for this element's value, which in this example is from 10:20am until 11:20am EST on 27 April 2002. In other words, the consumer of the SDE is being advised that after 1 hour the service data value is likely to no longer be valid, and therefore the client should query the service again for the SDE with the same **name** (**n1:MySDE**) to obtain a newer value of **n1:e1**.

The **availableUntil** does *not* refer to the service data value of the SDE, but rather to the availability of this named **serviceData** element itself. Prior to the declared **availableUntil** time, a client SHOULD be able to query the same service instance for an updated value of this named SDE. In this example, a client should be able to query the same service until 28 April 2002 10:20am EDT for the **serviceData** element named **n1:MySDE**, and receive a response with an updated copy of the **n1:e1** value. However, after that time, such a query MAY result in a response indicating that no such service data element exists. In other words, the consumer of the SDE is being advised that it can expect to be able to obtain an updated value of this named SDE for 1 day, but after that time the service may no longer have an SDE with the name **n1:MySDE**.

It is sometimes not sufficient for lifetime information of a SDE to refer only to the complete service data value. Rather, the value of a SDE may contain sub-elements with different lifetimes than those declared in the **serviceData** element. Any XML element contained within a **serviceData** element MAY use any combination of the **goodFrom**, **goodUntil**, and **availableUntil** attributes, assuming that the schema for that element allows for these extensibility attributes. Such attributes on sub-elements override the default values specified on parent elements. There are no constraints on the values of these attributes in the sub-elements, relative to

those specified in the parent elements, except that the ordering constraints between the effective `goodFrom`, `goodUntil` and `availableUntil` values for any element must be maintained.

In the above example, the lifetime attributes carried in the **serviceData** element provide default values for all children of that element. For example, the `n1:e2` element uses these default values, as described above. However, the `n1:e3` element overrides the **goodUntil** attribute, thus stating that its value (“def”) is only expected to be valid for 10 minutes, instead of 1 hour as is declared in the **serviceData** element. Such a situation might arise if a portion of a complex element changes more quickly than other portions of the element. Likewise, the `n1:e4` element overrides the **availableUntil**, thus stating that the `n1:e4` element may no longer exist within `n1:e1` after 10 hours. In other words, after 10 hours, a client that queries for the value of this **serviceData** element MAY be returned a `n1:e1` element that does not contain a `n1:e4` sub-element. This example, of course, assumes that the `MyType` schema allows for `n1:e4` to be an optional element, and thus be omitted from `n1:e1`.

It is RECOMMENDED that the XML schema for elements that are intended to be service data values allow all elements within their schema to be extended with these lifetime declaration properties, in order to allow for fine-grained lifetime declarations. However, since the **serviceData** element supports extensible properties, service data values that lack property extensibility can be enclosed with a **serviceData** element with the appropriate the lifetime declarations for that entire value.

Since a SDE MAY be an observation or “by-value copy” of service instance state or some other definitive source of the data, a processor of these properties MUST NOT assume that they necessarily reflect temporal aspects of that definitive source, unless otherwise specified (for example, in the semantic specification of a particular service data element). So the fact that a **serviceData** element has a particular **goodUntil** value does not necessarily imply that the underlying definitive source of that data will not change prior to that time.

If these attributes do not appear on an element, then the `goodFrom`, `goodUntil` and `availableUntil` properties are unknown.

4.4.5 The **wsdlLocation** attribute

Service data elements contain qnames that refer to WSDL-defined elements. For example, the **serviceData** element’s **name** attribute is the qname of its **serviceDataDescription**, and some service data values may contain qnames referring to various WSDL elements.

When a **serviceData** element appears within a WSDL document, the standard **wsdl:import** mechanism MUST be used when providing location information for resolving qname namespaces to WSDL documents.

However, when a **serviceData** element appears outside of a WSDL document (for instance, in a **serviceDataSet**, or in a response message to `FindServiceData`), the **gsdl:wsdlLocation** attribute MAY be used to provide location information for resolving qname namespaces to WSDL documents. This attribute is defined by the following schema definition:

```
<xsd:attribute name="wsdlLocation" type="xsd:string">
```

The content of the **gsdl:wsdlLocation** attribute mirrors that of **xsi:schemaLocation**: it MUST be a series of URI and URL pairs, where the first item of each pair MUST be a namespace URI and the second item MUST be a URL. The URL SHOULD resolve to a WSDL document that defines elements in that namespace.

When a service data element appears within a WSDL document, **gsdl:wSDLLocation** MUST NOT be used in place of the standard **wSDL:import** mechanism.

4.5 Interface Naming and Change Management

A critical issue in distributed systems is enabling the upgrade of services over time. This implies in turn that clients need to be able to determine when services have changed their interface and/or implementation. Here, we discuss this issue and some of the OGSA mechanisms, requirements, and recommendations that are used to address it.

4.5.1 The Change Management Problem

The semantics of a particular Grid service instance are defined by the combination of two things:

1. Its *interface specification*. Syntactically, a Grid service's interface is defined by its service description, comprising its **portTypes**, **operations**, **serviceDataDescriptions**, **messages**, and **types**. Semantically, the interface typically is defined in specification documents such as this one, though it may also be defined through other formal approaches.
2. The *implementation of the interface*. While expected implementation semantics may be implied from interface specifications, ultimately it is the implementation that truly defines the semantics of any given Grid service instance. Implementation decisions and errors may result in a service having behaviors that are ill-defined in and/or at odds with the interface specification. Nonetheless, such an implementation semantics may come to be relied upon by clients of that service interface, whether by accident or by design.

In order for a client to be able to reliably discover and use a Grid service instance, the client must be able to determine whether it is compatible with both of these two semantic definitions of the service. In other words, does the Grid service support the **portType** that the client requires? And does the implementation have the semantics that the client requires, such as a particular patch level containing a critical bug fix?

Further, Grid service descriptions will necessarily evolve over time. If a Grid service description is extended in a backward compatible manner, then clients that require the previous definition of the Grid service should be able to use a Grid service that supports the new extended description. Such backward compatible extensions might occur to the interface definition, such as through the addition of a new operation or service data description to the interface, or the addition of optional extensions to existing operations. Or, backward compatible extensions might occur through implementation changes, such as a patch that fixes a bug. For example, a new implementation that corrects an error that previously caused an operation to fail would generally be viewed as being backwards compatible.

However, if a Grid service description is changed in a way that is *not* backward compatible, a client MUST be able to recognize this as well. Again, this could be the result of incompatible changes to the interface or implementation of a Grid service. A bug fix that "fixes" an "erroneous" behavior that users have learned to take advantage of might not be considered backward compatible.

This discussion points to the need to be able to provide concise descriptions of both the interface and implementation of a Grid service, as well as to make unambiguous compatibility statements about Grid services that support different interfaces or implementations.

4.5.2 Naming Conventions for Grid Service Descriptions

In WSDL, each **portType** is globally and uniquely named via its qualified name—that is, the combination of the namespace containing the **portType** definition, and the locally unique name of the **portType** element within that namespace. In OGSA, our concern with change management leads us to require that all elements of a Grid service description **MUST** be immutable. The qname of a Grid service **portType**, **operation**, **message**, **serviceDataDescription** and underlying **type** definitions **MAY** be assumed to refer to one and only one WSDL/XSD specification. If a change is needed, a new **portType** **MUST** be defined with a new qname—that is, defined with a new local name, and/or in a new namespace.

Issue 24: Several people have commented on the need to loosen the immutability statement to say that a service description must be immutable once the service designer no longer has control of all possible clients of instances that use that description. The wording in this specification supports this interpretation because the **MUST**, **MAY**, etc. indicate what assumptions a compliant client or service implementation may make. A designer with full control of the client and hosting environment implementations can propagate changes without relying on features in the specification. However, this needs to be expressed more clearly in the text. (TBD: Resolve according to the minutes of 2002-09-06J.)

Issue 25: Some people have expressed the concern that the statement of immutability of service descriptions is too strict for small, backward compatible additions to a **portType**. One suggestion is to add some notion of a version number as a constant service data element. (TBD: Resolve according to the minutes of 2002-09-06J.)

TBD: Add text on service instance extension: **portType** of a running instance may change only to a type that extends the previous one. (Minutes of 2002-09-06J.)

4.6 Naming Grid Service Instances: Handles and References

Each Grid service instance is globally, uniquely, and for all time named by one or more *Grid Service Handles (GSH)*. However, a GSH is just a minimal name in the form of a URI, and does not carry enough information to allow a client to communicate directly with the service instance. Instead, a GSH must be *resolved* to a *Grid Service Reference (GSR)*. A GSR contains all information that a client requires to communicate with the service via one or more network protocol bindings.

Like any URI, a GSH consists of a *scheme*, followed by a string containing information that is specific to the scheme. The scheme indicates how one interprets the scheme-specific data to resolve the GSH into a GSR, within the bounds of the requirements defined below. A client **MAY** choose to resolve GSHs itself, or it **MAY** choose to outsource all resolution, for example, to a pre-configured service that implements the **HandleResolver portType** (see Section 7).

The format of the GSR is specific to the binding mechanism used by the client to communicate with the Grid service instance. For example, if an RMI/IIOP binding were used, the GSR would take the format of an IOR. If a SOAP binding were used, the GSR would take the form of a properly annotated WSDL document.

While a GSH is valid for the entire lifetime of the Grid service instance, a GSR may become invalid, therefore requiring a client to resolve the GSH into a new, valid GSR.

4.6.1 Grid Service Reference (GSR)

Grid service instances are made accessible to (potentially remote) client applications through the use of a Grid Service Reference (GSR). A GSR is typically a network-wide pointer to a specific

Grid service instance that is hosted in an environment responsible for its execution. A client application can use a GSR to send requests (represented by the operations defined in the WSDL **portType(s)** of the target service) directly to the specific instance at the specified (potentially network-attached) service endpoint identified by the GSR. In other words, the GSR supports the programmatic notion of passing Grid service instances "by reference". The GSR contains all of the information required to access the Grid service instance resident in its hosting environment over one or more communication protocol bindings. However, a GSR may be localized to a given client context or hosting environment, and the scope of portability for a GSR is determined by the binding mechanism(s) it supports.

The encoding of a Grid Service Reference may take many forms in the system. Like any other operation message part, the actual encoded format of the GSR "on the wire" is specific to the Web service binding mechanism used by the client to communicate with the Grid service instance. Below we define a WSDL encoding of a GSR that MAY be used by some bindings, but the use of any particular encoding is defined in binding specifications, and is therefore outside of the scope of this specification.

However, it is useful to elaborate further on this point here. For example, if an RMI/IIOP binding were used, the GSR would be encoded as a CORBA compliant IOR. If a SOAP binding were used, the GSR may take the form of the WSDL encoding defined below. This "on the wire" form of the Grid Service Reference is created both in the Grid service hosting environment, when references are returned as reply parameters of a WSDL defined operation, and by the client application or its designated execution environment when references are passed as input parameters of a WSDL defined operation. This "on the wire" form of the Grid Service Reference, passed as a parameter of a WSDL defined operation request message, SHOULD include all of the service endpoint binding address information required to communicate with the associated service instance over any of the communication protocols supported by the designated service instance, regardless of the Web service binding protocol used to carry the WSDL defined operation request message.

Any number of Grid Service References to a given Grid service instance MAY exist in the system. The lifecycle of a GSR MAY be independent of the lifecycle of the associated Grid service instance. A GSR is valid when the associated Grid service instance exists and can be accessed through use of the Grid Service Reference, but validity MAY only be detected by the client attempting to utilize the GSR. A GSR MAY become invalid during the lifetime of the Grid service instance. Typically this occurs because of changes introduced at the Grid service hosting environment. These changes MAY include modifications to the Web service binding protocols supported at the hosting environment, or of course, the destruction of the Grid service instance itself. Use of an invalid Grid Service Reference by a client SHOULD result in an exception being presented to the client.

When a Grid Service Reference is found to be invalid and the designated Grid service instance exists, a client MAY obtain a new GSR using the Grid Service Handle of the associated Grid service instance, as defined in Section 4.6.2. For convenience, the Grid Service Handle MAY be contained within a binding-specific implementation of the Grid Service Reference.

A binding-specific implementation of a Grid Service Reference MAY include an expiration time, which is a declaration to clients holding that GSR that the GSR SHOULD be valid prior to that time, and it MAY NOT be valid after the expiration time. After the expiration time, a client MAY continue to attempt to use the GSR, but SHOULD retrieve a new GSR using the GSH of the Grid service instance. While an expiration time provides no guarantees, it nonetheless is a useful hint in that it allows clients to refresh GSRs at convenient times (perhaps simultaneously with other

operations), rather than simply waiting until the GSR becomes invalid, at which time it must perform the (potentially time-consuming) refresh before it can proceed.

Mere possession of a GSR does not entitle a client to invoke operations on the Grid service. In other words, a GSR is not a capability. Rather, authorization to invoke operations on a Grid service instance is an orthogonal issue, to be addressed elsewhere.

4.6.1.1 WSDL Encoding of a GSR

It is RECOMMENDED that a WSDL document that encodes a GSR be the minimal information required to describe fully how to reach the particular Grid service instance. This information will commonly be just the WSDL **service** element, which in turn contains references (qnames) to elements in other namespaces of the other WSDL elements that are non-instance specific.

Issue 37: Should there be an element within wsdl:service to carry the GSH and/or expiration time?

4.6.2 Grid Service Handle (GSH)

Handles have the following properties:

1. A GSH MUST be a valid URI [RFC 2396].
2. A GSH MUST globally, uniquely, and for all time refer to the same Grid service instance. A GSH MUST NOT ever refer to more than one Grid service instance, whether or not they exist simultaneously. See Section 4.6.2.1 for the definition of “same Grid service instance.”
3. A Grid service instance MUST have at least one GSH.
4. A Grid service instance MAY have multiple GSHs that use different URI schemes.
5. A Grid service instance MAY have multiple GSHs that use the same URI scheme, if it is allowed by that URI scheme. However, the specification for a particular URI scheme MAY restrict this property by only allowing a single GSH within that URI scheme to a Grid service instance.
6. The **GridServiceHandles** service data element (Section 6.1) of a Grid service instance MUST contain only GSHs that refer to that instance. If two GSHs are contained in a Grid service instance’s **GridServiceHandles** SDE, then they MUST both refer to that same Grid service instance.
7. There MAY be multiple GSRs that refer to the same Grid service instance.
8. Multiple resolutions of the same GSH MAY result in different GSRs. A resolver MAY return different GSRs for the same GSH at different times, and it MAY return different GSRs to different clients that are resolving the same GSH.
9. A GSH MAY be unresolvable to a GSR before, during, or after the existence of the Grid service instance to which the GSH refers. However, the specification for a particular URI scheme MAY define a stronger quality of service for resolution. For example, a particular URI scheme MAY guarantee that resolution during the instance’s lifetime, and to a reliable statement of termination after the instance has terminated.
10. A client’s trust of a service may be handled in any way the client chooses, and therefore resolution of a GSH to a GSR MAY be either a trusted or an untrusted operation, depending on, for example, the configuration of the client, or the definition of the

resolution protocol. This specification permits, but does not require, a client to trust the resolution of a GSH to a GSR.

4.6.2.1 Service Instance Sameness

The interpretation of “same Grid service instance” is dependent upon the semantics of a service’s description. A service instance may be implemented in any way, as long as it obeys the semantics associated with its service description (i.e. the **portType** that the service instance implements).

For example, the implementation of a service MAY be distributed or replicated across multiple resources, as long as it obeys the semantics associated with its service description. A single GSH would be associated with this service, though that GSH may resolve to different GSRs referring to different resources, based on such factors as resource availability and utilization, locality of a client, client privileges, etc. Some service descriptions may require tight state coherency between the replicated implementations -- for example, the semantics of the service description may require that the service move through a series of well-defined states in response to a particular sequence of messages, thus requiring state coherence regardless of how GSHs are resolved to GSRs. However, other service descriptions may be defined that allow for looser consistency between the various members of the distributed service implementation.

4.6.3 XML types for handles, references and locators

Three XML type definitions are introduced so that handles and references may be introduced into typed message parts in the operation signatures of a WSDL service interface definition. The **gsdl:serviceReference** type contains a Grid Service Reference. The **gsdl:serviceHandle** type contains a Grid Service Handle. The **gsdl:serviceLocator** type contains zero or more GSHs and zero or more GSRs; it is used by various operations in this specification that may accept either a GSH or a GSR.

4.7 Grid Service Lifecycle

The lifecycle of any Grid service is demarked by the creation and destruction of that service. The actual mechanisms by which a Grid service is created or destroyed are fundamentally a property of the hosting environment, and as such are not defined in this document. There is nonetheless a collection of related **portTypes** defined in this specification that specify how clients may interact with these lifecycle events in a common manner. As we describe in subsequent sections:

- A client may request the *creation* of a Grid service by invoking the **createService** operation on a *Factory service*. (A service instance that implements a **portType** that extends the Factory **portType**.)
- A client may request the *destruction* of a Grid service via either client invocation of an *explicit* destruction operation request to the Grid service (see the **Destroy** operation, supported by the **GridService** **portType**: Section 6) or via a *soft-state* approach, in which (as motivated and described in [Grid Physiology]) a client registers interest in the Grid service for a specific period of time, and if that timeout expires without the service having received re-affirmation of interest from any client to extend the timeout, the service may be automatically destroyed. Periodic re-affirmation can serve to extend the lifetime of a Grid service as long as is necessary (see the **SetTerminationTime** operation in the GridService **portType**: Section 6).

In addition, a Grid service MAY support notification of lifetime-related events, through the standard notification interfaces defined in Section 8.

A Grid service MAY support soft state lifetime management, in which case a client negotiates an initial service instance lifetime when the Grid service is created through a factory (Section 9), and authorized clients MAY subsequently send SetTerminationTime (“keepalive”) messages to request extensions to the service's lifetime. If the Grid service termination time is reached, the server hosting the service MAY destroy the service, reclaim any resources associated with the service, and remove any knowledge of the service maintained in handle resolvers under its control.

Termination time MAY change non-monotonically. That is, a client MAY request a termination time that is earlier than the current termination time. If the requested termination time is before the current time, then this SHOULD be interpreted as a request for immediate termination.

A Grid service MAY decide at any time to extend its lifetime. A service MAY also terminate itself at any time, for example if resource constraints and priorities dictate that it relinquish its resources.

4.8 Common Handling of Operation Faults

TBD: We will define a small collection of base fault messages that may be used by the **portTypes** defined in this document. These faults will be described in this section in a subsequent version of this document. An XML schema of various XML types and wsdl:parts that define common fault messages for reuse amongst the operations defined in this specification and operations defined in domain-specific portTypes will appear with the final XML definition of Grid Services.

5 Grid Service Interfaces

A Grid service MUST implement the GridService portType, or a portType that extends the GridService portType. (See Section 6).

This specification defines a collection of common distributed computing patterns that are considered to be fundamental to OGSA. The embodiment of these patterns appears as WSDL **portTypes**. The collection of portTypes specified in this document is listed in Table 3.

The task for the designer of components within OGSA is to design **portTypes** that extend a combination of the **GridService portType**, other optional **portTypes** defined in this specification, and application or domain specific **portTypes**.

Table 3 Summary of the portTypes defined in this document

PortType Name	See Section	Description
GridService	6	encapsulates the root behavior of the component model
HandleResolver	7	mapping from a GSH to a GSR
NotificationSource	8.1	allows clients to subscribe to notification messages
NotificationSubscription	8.2	defines the relationship between a single NotificationSource and NotificationSink pair
NotificationSink	8.3	defines a single operation for delivering a notification message to the service instance that implements the operation
Factory	9	standard operation for creation of Grid service instances

Registration	10.2	allows clients to register and unregister registry contents
--------------	------	---

All of these portTypes (except of course for GridService) extend the GridService portType.

In the subsequent sections, all portTypes and serviceDataDescriptions are defined in the **gsdl** namespace.

6 The GridService PortType

In this section and those that follow, we describe the various standard **portTypes** that are defined by OGSA.

We start with the GridService **portType**, which **MUST** be implemented by all Grid services and thus serves as the base interface definition in OGSA. This **portType** is analogous to the base Object class within object-oriented programming languages such as Smalltalk or Java, in that it encapsulates the root behavior of the component model. The behavior encapsulated by the GridService **portType** is that of querying against the **serviceDataSet** of the Grid service instance, and managing the termination of the instance.

In Web services interface design, there is a choice to be made between document-centric messaging patterns and remote procedure call (RPC). Using a document-centric approach, the interface designer defines a loosely coupled interaction pattern wherein the API to the service is defined in terms of document exchange; both input and output are XML documents. This approach shifts the complexity of the interaction away from the API level and into the data format of the document exchange itself. This style tends to yield simpler, more flexible APIs. The RPC approach defines a specific, strongly-typed operation signature. This approach tends to produce less flexible API, but is often easier to map onto APIs of existing objects and can have better runtime performance.

Designers of Grid service interfaces also face the document-centric vs. RPC choice, and have attempted to take a middle road. The GridService **portType** provides several operations with typed parameters, but leaves considerable extensibility options within several of those parameters. Service data is then used to express what specific extensibility elements a particular service instance understands. Grid service designers are free to mix and match the document-centric and RPC approaches in the portTypes that they design to compose with those described here.

6.1 GridService PortType: Service Data Descriptions and Elements

The GridService **portType** includes **serviceData** elements conformant to the following **serviceDataDescription** elements¹:

```
<gsdl:serviceDataDescription
  name="PortType"
  element="gsdl:portTypeName"
  minOccurs="1"
  maxOccurs="1"
  mutability="constant">
  <wsdl:documentation>
    The qname of the portType implemented by this Grid service.
  </wsdl:documentation>
</gsdl:serviceDataDescription>
```

```

<gsdl:serviceDataDescription
  name="ServiceDataNames"
  element="gsdl:serviceName"
  minOccurs="0"
  maxOccurs="unbounded"
  mutability="mutable">
  <wsdl:documentation>
    A set of qnames, one for each service data element contained
    in this service instance. Note:the minOccurs corresponds to
    the number of serviceDataDescription elements in this
    GridService portType that are required (that have minOccurs > 0).
  </wsdl:documentation>
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription
  name="FactoryHandle"
  element="gsdl:serviceHandle"
  minOccurs="0"
  maxOccurs="1"
  mutability="constant">
  <wsdl:documentation>
    The Grid Service Handle to the factory that created
    this Grid service instance, if appropriate.
  </wsdl:documentation>
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription
  name="GridServiceHandles"
  element="gsdl:serviceHandle"
  minOccurs="1"
  maxOccurs="unbounded"
  mutability="append">
  <wsdl:documentation>
    The Grid Service Handles of this Grid service instance.
    It is possible to have multiple handles, for example one
    for each handle scheme that might be deployed.
  </wsdl:documentation>
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription
  name="GridServiceReferences"
  element="gsdl:serviceReference"
  minOccurs="1"
  maxOccurs="unbounded"
  mutability="mutable">
  <wsdl:documentation>
    A set of Grid Service References to this Grid service instance.
    One service data value element MUST be the WSDL representation
    of the GSR. Other service data value elements may represent
    other forms of the WSDL.
  </wsdl:documentation>
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription
  name="QueryExpressionTypes"
  element="gsdl:queryExpressionType"

```

```

        minOccurs="1"
        maxOccurs="unbounded"
        mutability="append">
<wsdl:documentation>
    A set of qnames of element declarations. Any conforming
    element MAY be used by the client as FindServiceData's
    QueryExpression parameter.
</wsdl:documentation>
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription
    name="TerminationTime"
    element="gsdl:terminationTime"
    minOccurs="1"
    maxOccurs="1"
    mutability="mutable">
<wsdl:documentation>
    The earliest and latest termination times for the grid service
    (see 6.2.2).
</wsdl:documentation>
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription
    name="CurrentTime"
    element="gsdl:currentTime"
    minOccurs="1"
    maxOccurs="1"
    mutability="mutable">
<wsdl:documentation>
    The current time as known to this service. This value can be
    used to gauge the clock skew of this service relative to a
    client.
</wsdl:documentation>
</gsdl:serviceDataDescription>

```

In addition, the **GridService portType** defines the following initial set of service data value elements:

```

<gsdl:serviceData name="gsdl:QueryExpressionTypes">
  <gsdl:queryExpressionType>
    gsdl:queryByServiceDataName
  </gsdl:queryExpressionType>
</gsdl:serviceData>

```

6.2 GridService PortType: Operations and Messages

Issue 29: Do we need an additional operation on GridService that allows a client to reliably determine if a GSH (or perhaps two GSHs) refers to particular instance?

6.2.1 GridService :: FindServiceData

Query the service data.

Input

- *QueryExpression*: The query to be performed. This is an extensible parameter, which **MUST** conform to an element declaration denoted by one of the *QueryExpressionTypes*. Note: The service infers what to do based on the tag of the root element of this argument.

Output

- *Result*: The result of the query. The format of this result is dependent upon the *QueryExpression*.

Fault(s)

- TBD.

Every Grid service instance **MUST** support *QueryExpressions* conforming to **queryByServiceDataName** as defined in 6.2.1.1. A Grid service instance **MAY** support other *QueryExpressions*

The list of query expression types supported by a Grid service instance is expressed in the instance's **QueryExpressionTypes** service data element. Therefore, a client can discover the query expression types supported by a service instance by performing a **FindServiceData** request on the instance, using the **queryByServiceDataName** element with name="gsdl:QueryExpressionTypes".

The service data that is available for query by a client **MAY** be subject to policy restrictions. For example, some service data elements **MAY** not be available to some clients, and some service data value elements within a SDE **MAY** not be available to some clients.

6.2.1.1 queryByServiceDataName

A **queryByServiceDataName** results in the service data element that has the QName as specified by the **name** property (which **MUST** be among the *ServiceDataNames* (see 6.1) contained in this service instance).

The non-normative grammar of this type is:

```
<gsdl:queryByServiceDataName name="qname" />
```

The *FindServiceData* operation's *Result* output parameter for a **queryByServiceDataName** query **MUST** be the **serviceData** element that has the requested **serviceDataName**.

Issue 53: We should consider either extending *queryByServiceDataName* or define a new *queryExpression: queryByServiceDataNameCollection* to allow multiple names to be specified on the query. That way, a client can with one operation query the simultaneous values of multiple SDEs.

6.2.1.2 queryByXPath

Issue 30: Define the XPath *QueryExpressionType*, *QueryExpression* schema, and the result format. Support for this *QueryExpressionType* is optional.

6.2.1.3 queryByXQuery

Issue 31: Define the XQuery *QueryExpressionType*, *QueryExpression* schema, and the result format. Support for this *QueryExpressionType* is optional.

6.2.2 GridService :: RequestTerminationAfter

Request that the termination time of this service be changed. The request specifies the earliest desired termination time. Upon receipt of the request, the service **MAY** adjust its termination

time, if necessary, based on its own policies and the requested time. Upon receipt of the response, the client SHOULD discard any responses that have arrived out of order, based on the CurrentTimestamp in the response.

Input:

- *TerminationTime*: The earliest termination time of the Grid service that is acceptable to the client. A value in the past indicates that the client no longer cares about the earliest termination time. The special value “infinity” means that the client requests that the service continue to exist indefinitely.

Output:

- *CurrentTimestamp*: The time at which the Grid service handled the request.
- *CurrentTerminationTime*: The service's currently planned termination time (as described below).

Fault(s):

-

The CurrentTerminationTime returned by this operation is of the form given by the following non-normative grammar:

```
<terminationTime
  after="gsdl:extendedDateTime"
  before="gsdl:extendedDateTime" />
```

This element describes the service's current plans for termination. The “**after**” attribute gives the earliest time at which the service plans to terminate. A value in the past indicates that the service may terminate at any time. The special value “infinity” indicates that the service plans to exist indefinitely. The “**before**” attribute gives the latest time at which the service plans to terminate. A value in the past indicates that the service is trying to terminate. The special value “infinity” indicates that the service has no plans to terminate. The maximum time MUST be greater than or equal to the minimum time.

6.2.3 GridService :: RequestTerminationBefore

Request that the termination time of this service be changed. The request specifies the latest desired termination time. Upon receipt of the request, MAY adjust its termination time, if necessary, based on its own policies and the requested time. Upon receipt of the response, the client SHOULD discard any responses that have arrived out of order, based on the timestamp in the response.

Input:

- *TerminationTime*: The latest termination time of the Grid service that is acceptable to the client. A time in the past indicates a desire that the service terminate as soon as possible. The special value “infinity” indicates that the client no longer cares about a maximum termination time.

Output:

- *CurrentTimestamp*: The time at which the Grid service handled the request.
- *CurrentTerminationTime*: The service's currently planned termination time (as described in 6.2.2).

Fault(s):

-

6.2.4 GridService :: Destroy

Explicitly request destruction of this service. Upon receipt of an explicit destruction request, a Grid service **MUST** either initiate its own destruction and return a response acknowledging the receipt of the destroy message; or ignore the request and return a fault message indicating failure. Once destruction of the Grid service is initiated, the service **SHOULD NOT** respond to further requests. Following a successful Destroy() call, the client **SHOULD NOT** rely on the existence of the service.

Input:

- None

Output:

- None, indicating that the destroy has been initiated. Some services may strengthen this to mean that the service destruction is effectively complete.

Fault(s):

- *ServiceNotDestroyed*, indicating that the service chose not to initiate self-destruction.

7 The HandleResolver PortType

A *handle resolver* is a Grid service instance that implements the HandleResolver **portType**.

Issue 17: The authoring team is divided on the recommendations regarding the role of the resolver protocol vs the use of HandleResolver. Should there be language such as: "Clients **SHOULD** use one or more HandleResolver services to resolve GSHs to GSRs. The handle resolver protocols discussed in this section **SHOULD NOT** normally be used by clients, but **SHOULD** be used by HandleResolver services only."

Each GSH scheme defines a particular *resolver* protocol for resolving a GSH of that scheme to a GSR. Some schemes, such as the http and https schemes defined in Section 4.6.2, **MAY** not require the use of a HandleResolver service, as they are based on some other resolver protocol. However, there are two situations where a Grid service based resolver protocol **MAY** be used, and which therefore motivates the definition of a standard **HandleResolver portType**. First, a GSH scheme **MAY** be defined that uses the **HandleResolver** as a fundamental part of its resolver protocol, where the GSH carries information about to which **HandleResolver** service instance a client should send resolution requests. Second, in order to avoid placing undo burden on a client by requiring it to directly speak various resolver protocols, a client instead **MAY** be configured to outsource any GSH resolutions to a third party **HandleResolver** service. This outsourced handle resolver **MAY** in turn speak the scheme-specific resolver protocols directly. Both of these situations are addressed through the definition of the HandleResolver **portType**.

Various handle resolvers may have different approaches as to how they are populated with GSH to GSR mappings. Some handle resolvers may be tied directly into a hosting environment's lifetime management services, such that creation and destruction of instances will automatically add and remove mappings, through some out-of-band, hosting-environment-specific means. Other handle resolver services may implement the Registration **portType**, such that whenever a service instance registers its existence with the resolver, that resolver queries the GridServiceHandles and GridServiceReferences service data elements of that instance to construct its mapping database. Other handle resolver services may implement a custom

registration protocol via a custom **portType**. But in all of these cases, the **HandleResolver** **portType** MAY be used to query the resolver service for GSH to GSR mappings.

7.1 **HandleResolver PortType: Service Data Descriptions**

The **HandleResolver portType** includes **serviceData** elements conformant to the following **serviceDataDescription** elements:

```
<gsdl:serviceDataDescription
  name="HandleResolverSchemes"
  element="xsd:anyURI"
  minOccurs="0"
  maxOccurs="unbounded"
  mutability="mutable">
  <wsdl:documentation>
    A set of URIs that correspond to the handleResolver schemes that
    the HandleResolver implements.
  </wsdl:documentation>
</gsdl:serviceDataDescription>
```

7.2 **HandleResolver PortType: Operations and Messages**

7.2.1 **HandleResolver :: FindByHandle**

Returns a **serviceLocator**, which contains one or more Grid Service References for a Grid Service Handle.

Input

- *Handle*: A Grid Service Handle.
- *GSRSet*: (optional) a set of one or more GSRs that the client already possesses that are not satisfactory for some reason. This is a hint from the client that these existing references should not be returned in response to this message.

Output

- *Locator*: A service locator containing one or more Grid Service References, and zero or more alternate Grid Service Handles to the same service instance.

Fault(s)

- *InvalidHandle*, indicating that the handle violates the syntax of its URI scheme.
- *NoAdditionalReferencesAvailable*, indicating that the resolver service cannot return a GSR that is not already contained in the *GSRSet* input parameter.
- *NoReferencesAvailable*, indicating that the resolver service is unable to return a GSR for the input handle, regardless of the *GSRSet* input parameter.
- *Redirection*, indicating an alternate handle resolver to which the client MAY direct the request. The GSH of the alternate handle resolver is returned with the fault.
- *NoSuchService*, indicating that either there was never a service with this handle, or the service with this handle has terminated. This fault MAY only be applicable to some URI schemes.
- *NoSuchServiceStarted*, indicating that there was never a service with this handle.

- *ServiceHasTerminated*, indicating that the service with this handle has terminated. This fault MAY only be applicable to some URI schemes.
- *TemporarilyUnavailable*, indicating that the handle refers to a valid service, but that it cannot be resolved to a valid reference at this time, though it may be resolvable later. This fault optionally returns a time at which the service MAY be available. This fault MAY only be applicable to some URI schemes.

8 Notification

Issue 57: We need to discuss this mechanism in more depth. We have gotten several requests for "Topics" and "events" and it is not clear we can model these as SDEs. This is a place holder and I suspect more commentary text will be added to flesh this concern out.

The purpose of notification is to deliver interesting messages from a notification source to a notification sink, where:

- A *notification source* is a Grid service instance that implements the NotificationSource **portType**, and is the sender of notification messages. A source MAY be able to send notification messages to any number of sinks.
- A *notification sink* is a Grid service instance that receives notification messages from any number of sources. A sink MAY implement the DeliverNotification operation of the NotificationSink **portType**, which allows it to receive notification messages of any type. Alternatively, a sink MAY implement a *specialized notification delivery operation* from a different **portType**, where that operation is a specialization of the DeliverNotification operation. A specialized delivery operation MAY only accept a subset of the types of messages that the general DeliverNotification operation can accept, and like DeliverNotification is an input-only operation (i.e. it does not return a response).
- A *notification message* is an XML element sent from a notification source to a notification sink. The XML type of that element is determined by the subscription expression.
- A *subscription expression* is an XML element that describes what messages should be sent from the notification source to the notification sink. The subscription express also describes when messages should be sent, based on changes to values within a service instance's **serviceDataSet**.
- In order to establish what and where notification messages are to be delivered, a *subscription* request is issued to a source, containing a subscription expression, the **serviceLocator** of the notification sink to which notification messages are to be sent, the portType and operation name of the specialized notification delivery operation to which notification messages should be sent, and an initial lifetime for the subscription.
- A subscription request causes the creation of a Grid service instance, called a *subscription*, which implements the NotificationSubscription **portType**. This **portType** MAY be used by clients to manage the (soft-state) lifetime of the subscription, and to discover properties of the subscription.

This notification framework allows for either direct service-to-service notification message delivery, or for the ability to integrate various intermediary delivery services. Intermediary delivery services might include: messaging service products commonly used in the commercial world, message filtering services, message archival and replay services, etc.

8.1 The NotificationSource PortType

The NotificationSource **portType** allows clients to subscribe to notification messages from the Grid service instance that implements this **portType**.

8.1.1 NotificationSource PortType: Service Data Descriptions and Elements

The NotificationSource **portType** includes **serviceData** elements conformant to the following **serviceDataDescription** elements:

```
<gsdl:serviceDataDescription
  name="NotifiableServiceDataNames"
  element="gsdl:serviceDataName"
  minOccurs="0"
  maxOccurs="unbounded"
  mutability="mutable">
  <wsdl:documentation>
    A set of qualified names of service data elements to
    which a client MAY subscribe for notification of changes.
  </wsdl:documentation>
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription
  name="SubscriptionExpressionTypes"
  element="gsdl:subscriptionExpressionType"
  minOccurs="1"
  maxOccurs="unbounded"
  mutability="append">
  <wsdl:documentation>
    A set of qnames of element declarations. Any conforming
    element MAY be used by the client as Subscribe's
    SubscriptionExpression parameter.
  </wsdl:documentation>
</gsdl:serviceDataDescription>
```

The NotificationSource portType would also include the following initial service data value elements:

```
<gsdl:serviceData
  name="gsdl:SubscriptionExpressionTypes">
  <gsdl:subscriptionExpressionType
    type="gsdl:subscribeByServiceDataName"/>
</gsdl:serviceData>
```

8.1.2 NotificationSource PortType: Operations and Messages

8.1.2.1 NotificationSource :: Subscribe

Subscribe to be notified of subsequent changes to the target instance's service data. This operation creates a Grid service subscription instance, which MAY subsequently be used to manage the lifetime and discovery properties of the subscription.

Input:

- *SubscriptionExpression*: The subscription to be performed. This is an extensible parameter, which **MUST** conform to an element declaration referred to by one of the *QueryExpressionTypes* qnames.
- *Sink*: The **serviceLocator** of the notification sink to which messages will be delivered. This locator **MAY** be to some other service than the one that is issuing this subscription request, thus allowing for third-party subscriptions.
- *SpecializedNotificationDeliveryOperation* (optional): The name of the **operation**, and the qname of the **portType** in which that operation is defined, to be used by the notification source when delivering messages to the notification sink. The operation signature **MUST** be the same as, or a specialization of, the `NotificationSink::DeliverNotification` operation. If this parameter is not specified, then it defaults to the “DeliverNotification” **operation** name that is defined in the `gsdl:NotificationSink` **portType**.
- *ExpirationTime*: The initial time at which this subscription instance should terminate, and thus notification delivery to this sink be halted. Normal GridService lifetime management operations **MAY** be used on the subscription instance to change its lifetime.

Output:

- *SubscriptionInstanceLocator*: A **serviceLocator** to the subscription instance that was created to manage this subscription. This subscription instance **MUST** implement the `NotificationSubscription` **portType**.

Fault(s):

Every Grid service instance that implements the `NotificationSource` `portType` **MUST** support a *SubscriptionExpressionType* of **subscribeByServiceDataName** as defined in 8.1.2.1.1. A Grid service instance **MAY** support other *SubscriptionExpressionTypes*.

The list of subscription expression types supported by a Grid service instance is expressed in the instance’s **SubscriptionExpressionTypes** service data element. Therefore, a client can discover the subscription expression types supported by a service instance by performing a **FindServiceData** request on the instance, using a **queryByServiceDataName** element, which contains the name “`gsdl:SubscriptionExpressionTypes`”.

The service data that is available for subscription by a client **MAY** be subject to policy restrictions. For example, some service data elements **MAY** not be available to some clients, and some service data value elements within a SDE **MAY** not be available to some clients.

8.1.2.1.1 *subscribeByServiceDataName*

A **subscribeByServiceDataName** results in notification messages being sent whenever the named service data element changes.

The non-normative grammar of this type is:

```
<gsdl:subscribeByServiceDataName
  name="xsd:qname"
  minInterval="xsd:duration"?
  maxInterval=("xsd:duration" | "unbounded")? >
</gsdl:subscribeByServiceDataName>
```

The **minInterval** property specifies the minimum interval between notification messages, expressed in `xsd:duration`. If this property is not specified, then the notification source **MAY**

choose this value. A notification source MAY also reject a subscription request if it cannot satisfy the minimum interval requested.

The **maxInterval** property specifies the maximum interval between notification messages, expressed in xsd:duration. If this interval elapses without a change to the named service data element's value, then the source MUST resend the same value. When the value is "unbounded" the source need never resend a service data value if it does not change. If this property is not specified, then the notification source MAY choose this value.

For a **subscribeByServiceDataName** subscription, the type of the notification message sent from the notification source to the notification sink MUST be the **serviceData** element that has the requested **serviceDataName**.

8.2 The NotificationSubscription PortType

A subscription for notification causes the creation of a Grid service *subscription* instance, which MUST implement the NotificationSubscription **portType**. This instance MAY be used by clients to manage the lifetime of the subscription, and discover properties of the subscription.

8.2.1 NotificationSubscription PortType: Service Data Descriptions

The NotificationSubscription **portType** includes **serviceData** elements conformant to the following **serviceDataDescription** elements:

```
<gsdl:serviceDataDescription
  name="SubscriptionExpression"
  element="gsdl:subscriptionExpression"
  minOccurs="1"
  maxOccurs="1"
  mutability="constant">
  <wsdl:documentation>
    The current subscription expression managed by this
    subscription instance.
  </wsdl:documentation>
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription
  name="SinkLocator"
  element="gsdl:serviceLocator"
  minOccurs="1"
  maxOccurs="1"
  mutability="constant">
  <wsdl:documentation>
    The Grid Service Locator of the Notification sink to
    which this subscription is delivering messages.
  </wsdl:documentation>
</gsdl:serviceDataDescription>
```

8.2.2 NotificationSubscription PortType: Operations and Messages

None.

8.3 The NotificationSink PortType

A notification sink **portType** defines a single operation for delivering a notification message to the service instance that implements the operation. **Issue 58: Need to be careful about this it is**

overly restrictive. If you allow the subscribe operation to be passed a GSR (read WSDL) then the “Sink” could be any WebService not restricted to just GridServices. I think this is a good thing that lets “Sink”s be lighter weight

8.3.1 NotificationSink PortType: Service Data Descriptions

None.

8.3.2 NotificationSink PortType: Operations and Messages

8.3.2.1 NotificationSink :: DeliverNotification

Deliver message to this service.

Input:

- *Message*: An XML element containing the notification message. The content of the message is dependent upon the notification subscription.

Output:

The service does not reply to this request.

Fault(s):

8.4 Integration With Notification Intermediaries

While the NotificationSource and NotificationSink define how notification messaging is performed between two parties, these same **portTypes** can be used in various combinations to allow for third-party services to intermediate the notification process.

For example, an intermediary notification service may implement the NotificationSink **portType** in order to receive notification messages from some other sources, as well as the NotificationSource **portType** to send notifications to other subscribing sinks. The intermediary may simply forward the notification messages on to subscribers, or it may transform them in various ways by making service data elements available to subscribers that are different than SDEs of the original notification source. Intermediary notification sources are generally characterized by the fact that their **serviceData** elements have **originator** properties that refers to other service instances, rather than to themselves.

Intermediary notification services may be used for a variety of purposes, including:

- To provide for a notification source service that has a lifetime that is independent from that of the notification source service that originally generated the message.
- To filter, modify, aggregate, and/or archive notification messages from other sources.
- To represent third party messaging services, which may transport notification messages with different delivery protocols, semantics, and/or qualities of service.

The third purpose, integrating messaging service products, deserves further explanation. Such messaging service products can be exploited in this framework by:

1. Defining an intermediary messaging service instance that implements both the NotificationSource and NotificationSink portTypes, as well as possibly other portTypes for managing the behavior of the messaging service product.
2. This intermediary messaging service instance can then subscribe to various notification source. Note that the client issuing the subscription request need not be the same Grid

service instance as the notification sink designated in the subscription request to receive notification messages. This property allows for clients to stitch together notification message paths, without being directly in those paths.

3. The intermediary messaging service instance can advertise various notification topics service data elements for which it produces notification messages, relating to any incoming notification messages it receives via its sink interface. For example, for any notification message that it receives through its sink interface, it may resend it to subscribers with a particular quality of service.
4. The intermediary messaging service instance may have its own efficient, scalable, message distribution network, thus allowing the incoming message to be efficiently delivered to a large number of subscribing sinks. Or it may guarantee delivery of the notification message for some period of time, even in the face of various failures. Or it may distribute incoming notification messages to sinks in a round-robin fashion, rather than sending all notification messages to all sinks. The possible behaviors that the intermediary messaging service instance can introduce to the notification message delivery are limitless.
5. The intermediary messaging service instance, the originating notification source service, and the final notification sink service may all implement a specialized network protocol binding to optimize the transmission of the notification messages. For example, if the intermediary messaging service instance represents a particular message service product with its own custom protocol, implementing that protocol as a Grid service network protocol binding allows the integration of this product and its protocols, without requiring different interfaces or models to be imposed on the sources and sinks.

9 The Factory portType

From a programming model perspective, a *factory* is an abstract concept or pattern. A factory is used by a client to create an instance of a Grid service. A client invokes a create operation on a factory and receives as response a serviceLocator for the newly created service. This specification defines one approach to realizing the factory pattern as a Grid service. OGSA uses a document-centric approach to define the operations of the basic factory. Service providers can, if they wish, define their own factories with specifically typed operation signatures.

In OGSA terms, a factory is a Grid service that **MUST** implement the Factory **portType**, which provides a standard WSDL operation for creation of Grid service instances. A factory **MAY** of course also implement other **portTypes** (in addition to the required GridService **portType**), such as:

- Registration (Section10), which allows clients to inquire of the factory as to what Grid service instances created by the factory are in existence.

Upon creation by a factory, the Grid service instance **MUST** be registered with, and receive a GSH from, a handle resolution service (see Section 7). The method by which this registration is accomplished is specific to the hosting environment, and is therefore outside the scope of this specification.

Issue 5: Consider adding another factory related portType that allows for management of the set of portTypes that a factory may create. Perhaps allow for downloading of code into the factory when adding a service.

9.1 Factory PortType: Service Data Descriptions

The Factory **portType** includes **serviceData** elements conformant to the following **serviceDataDescription** elements:

```
<gsdl:serviceDataDescription
  name="CreatesPortTypes"
  element="gsdl:portTypeName"
  minOccurs="1"
  maxOccurs="unbounded"
  mutability="append"
  <wsdl:documentation>
    QNames to most derived portTypes of service instances that can
    be created by this Factory.
  </wsdl:documentation>
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription
  name="CreationInputTypes"
  element="gsdl:creationExpressionType"
  minOccurs="0"
  maxOccurs="unbounded"
  mutability="append"
  <wsdl:documentation>
    Qnames of XML elements supported by this Factory for the
    ServiceParameters argument of the CreateService operation.
  </wsdl:documentation>
</gsdl:serviceDataDescription>
```

9.2 Factory PortType: Operations and Messages

9.2.1 Factory :: CreateService

Create a new Grid service instance. Note that to support soft state lifetime management (Section 4.7), a client may specify an initial termination time, within a window of earliest and latest acceptable initial termination times. The factory selects an initial termination time within this window, and returns this to the client as part of its response to the creation request. Additionally, the factory returns the maximum lifetime extension that clients can subsequently request of this new Grid service instance. Alternatively, the Grid service creation request may fail if the requested termination time is not acceptable to the factory.

Input

- *TerminationTime* (optional): The earliest initial termination time of the Grid service instance that is acceptable to the client.
- *ServiceParameters* (optional): An XML document that is specific to the factory and the services that it creates.

Output

- *ServiceLocator*: A **serviceLocator** to the newly created Grid service instance.
- *ServiceTimestamp*: The time at which the Grid service was created.
- *CurrentTerminationTime*: The Grid service's currently planned termination time.
- *MaximumExtension*: The maximum extension that the Grid service will currently allow a client to request of its termination time.

- *ExtensibilityOutput* (optional): An XML extensibility element that is specific to the factory and the services that it creates.

Fault(s):

Issue 19: Can we reduce the number of output parameters in `Factory::CreateService` by moving many of them into service data of the created instance?

10 Registration

Issue: 52 It is becoming more clear that the `Registration portType` is overloaded. I recommend that we refactor `Registration portType` into two (or more) `portTypes`. There are several models being debated.

A *registry* is a Grid service that maintains a collection of Grid Service Handles, with policies associated with that collection. Clients may query the registry to discover what services are available.

A registry implements the `Registration portType`, in order to allow clients to register and unregister registry contents. Because a registry is a Grid service, it must also implement the `GridService portType`. A registry MAY implement custom `portType` that define service data elements that are structured to support particular types of queries against the registry. The **FindServiceData** (see Section 6.2.1) operation provides rich query interface against the contents of the registry that is maintained in service data. A registry's **FindServiceData** operation SHOULD support the XPath query language, and MAY support other query languages. A registry SHOULD implement the **NotificationSource portType** (Section 8), in order to support notification of registry existence and changes in registry contents.

A Grid service instance MAY be a member of any number of registries, and for any portion of the service's lifetime.

10.1 WS-Inspection Document

The registry makes available a WS-Inspection document [WSIL Overview] to aid in discovery of the services in that registry. This document contains information about any Grid service that has been registered with the registry.

This WS-Inspection document can be retrieved by a client via the `FindServiceData` operation.

The registry's WS-Inspection document MAY have the following properties:

1. WS-Inspection WSDL service description elements that refer other Grid services (including other registries), where the location values are the GSHs of the services.
2. WS-Inspection link elements that refer to registry services, where the location values are the GSHs of services.
3. any other valid WS-Inspection element.

Issue 20: Should a WS-Inspection document be required of all registries? Or might a registry just use the `RegisterService` operation, define its own service data to represent the contents of the registry, and ignore the WS-Inspection document entirely? In other words, should the WS-Inspection document be given a `minOccurs="0"`, or separated out into a different `portType` entirely?

10.2 The Registration portType

The *Registration portType* allows clients to register and unregister registry contents.

10.2.1 Registration PortType: Service Data Descriptions

The following contains the **serviceDataDescription** elements associated with the *Registration portType*:

```
<gsdl:serviceDataDescription
  name="RegistrationExtensibilityTypes"
  element="gsdl:registrationExtensibilityType"
  minOccurs="1"
  maxOccurs="unbounded"
  mutability="append">
  <wsdl:documentation>
    A set of qnames of element declarations. Any conforming
    element MAY be used by the client as RegisterService's
    extensibility parameter.
  </wsdl:documentation>
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription
  name="RegistrationContentNames"
  element="gsdl:serviceName"
  minOccurs="0"
  maxOccurs="unbounded"
  mutability="append">
  <wsdl:documentation>
    A set of names of serviceData elements relating to
    registry content.
  </wsdl:documentation>
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription
  name="GridServiceRegistryWSInspection"
  element="wsil:inspection"
  minOccurs="0"
  maxOccurs="1"
  mutability="mutable"
  <wsdl:documentation>
    A WS-Inspection document containing all of the Grid services in
    this registry. We expect that specializations of registries that
    take this registration port type and combine it with other
    mechanisms to do discovery (for example a specialization that
    finds discovery such as "find by portType"). This portType forms
    the base assumption that an inspection document is available.
  </wsdl:documentation>
</gsdl:serviceDataDescription>
```

10.2.2 Registration PortType: Operations and Messages

10.2.2.1 Registration :: RegisterService

Add or atomically update a Grid Service Handle to the registry.

Input

- *Locator*: The Grid Service locator of the service to register. If the registry already contains a registration that matches this parameter, this operation is to be treated as an "update" to the registration information.

- *Extensibility*: An XML fragment to be inserted as an extensibility element for this service. This field may be used, for example to register a UDDI service description for the service as well. This element **MUST** conform to an element declaration referred to by one of the `RegistrationExtensibilityTypes`. Note the **gsdl:emptyExtensibility** element `qname` can be used to make this parameter effectively optional.

Output:

- None, acknowledging receipt of the `RegisterService` call.

Fault(s):

-

It is worth noting that the registry has the freedom to interpret registrations of other Registration Grid services in one of two ways. Consider the case where Registry A receives a register operation for Registry B. Registry A can choose to treat B simply as another Grid service, generating a WS-Inspection **service** element based on the GSH for B, or it can be treat B as a WS-Inspection *link* element.

10.2.2.2 Registration :: UnregisterService

Remove a Grid Service Handle from the registry.

Input:

- *Handle*: The Grid Service Handle of the service to remove.

Output:

- None, acknowledging receipt of the `UnregisterService` call.

Fault(s):

- Unregistration operation failed.

11 Security Considerations

This specification defines the abstract interaction between a Grid service and clients of that service. While it is assumed that such interactions must be secured, the details of security are out of scope of this specification. Instead, security should be addressed in related specifications that defined how the abstract interactions are bound to specific communication protocols, and to specific programming environments.

12 Change Log**12.1 Draft 1 (2/15/2002) → Draft 2 (6/13/2002)**

- Improved introduction to Section 4, “The Grid Service, and reordered the subsections to make it flow better.
- Added Section 4.2, “Service Description and Service Instance”, containing an explanation of service description vs service instance.
- Added/rewrote “Service Data” section (4.3) including: cleaned up **serviceData** container; moved lifetime declarations out to an extensibility element that can be included on any XML element; introduced schema to be able include service data declarations into the WSDL service description.

- Changed tables containing service data declarations to use correct XML elements that conform to the new **serviceDataDescription** element
- Moved description of **instanceOf** to be part of the WSDL GSR description, since it is a sub-element of the WSDL **service** element, which is part of the WSDL GSR.
- Removed old Section 5, “How portType Definitions are Presented”. This was subsumed by the rewrite of Section 4, including the new service data specification.
- Removed all primary key material, including old Section 10, and references to it from the Factory discussion.
- Simplified the schema for serviceType.
- Added Section 11, Change Log.

12.2 Draft 2 (6/13/2002) → Draft 3 (07/17/2002)

- Changed draft to assume new features in WSDL v1.2 draft 1, including **serviceType** and an open extensibility model.
- Added serviceType reuse/extension.
- Modified notion of Handle to be a URI, reflected changes in GSR and HandleResolver (previously called HandleMap) discussion. Introduced resolver protocols for the http and https GSH schemes.
- Substantially changed “Service Data” section (4.3), primarily to cleanup and plug holes in service data descriptions and elements, particularly around naming and typing. Changed various portType descriptions to reflect this change to service data.
- Added section “Modeling Time in OGSA”
- Overhauled the notification section, to completely integrate with service data, and to provide a “push model” that parallels the FindServiceData “pull model”.
- Renamed Registry portType to Registration, and did some cleanup on the section.
- Introduced gsd:**serviceLocator**, which is an XML schema type that can be either a GSH or GSR. Changed various GSH and GSR argument to use this type.
- Renamed “Terminology and Abbreviations” section to “Notational Conventions”. In this section, added a table of namespace prefixes used in throughout the document, and cleaned up the rest of the section.
- Added inline “Issue” that need to be resolved, with numbers that refer to the GGF OGSI working group bugzilla database.

12.3 Draft 3 (07/17/2002) → Draft 4 (10/1/2002)

- Reformatted for GGF document compliance.
- Updated and reformatted references.
- Added “Security Considerations” section.
- **TBD:** “Action items” from minutes.
- Changed (in 4.4.2) the “type” attribute of SDD to be “element”, and reference an XSD element declaration, not a type definition, resolving issue 32. The defaults for minOccurs and maxOccurs were changed to 1 to be consistent with XML schema.

- Added text to 4.4.3 about whether minOccurs>0 makes sense in light of authorization (issue 40, M2002-08-28).
- Added non-normative text to Service Data 4.4 to help better explain/motivate the concept.
- Added a definition of **gsdl:wSDLLocation** (See minutes 2002-09-06E).
- In 4.5.2, changed “gsdl:serviceType” to “gsdl:extends”.
- Added **gsdl:serviceHandle** and **gsdl:serviceReference** types (4.6.3). Changed serviceLocator to contain sets of these.
- Added a CurrentTime SDD to GridService (M2002-09-06F).
- Removed the URI argument to GridService::FindServiceData (6.2.1). The tag of the QueryExpression argument (a QName) is used to determine the query to be performed. Similarly, removed the URI argument to NotificationSource::Subscribe (8.1.2.1). (Issue 33, M2002-09-06A)
- Changed SetTerminationTime to be two operations (see, 6.2.2 and 6.2.31 – see minutes 2002-08-21) and modified them according to M2002-09-06F. Also modified the TerminationTime SDD to reflect both times.
- Modified Destroy according to 2002-09-06H minutes.
- Added to the Registration portType (10.2.1) an SDD to indicate the allowed extensibility arguments and an SDD to indicate registration content SDEs (2002-08-28 minutes). Also, the WSIL SDE was made optional.
- Modified RegisterService (10.2.2.1) according to 2002-09-06B minutes.
- In addition to the above, the descriptions of the following issues were removed:

Issue(s)	Location	Resolution
13	6.2.2 ?	See minutes 2002-09-06F
14	6.2.2	See minutes 2002-08021; called “infinity”
15	6.2.2	See minutes 2002-08-21
16	6.2.2	See minutes 2002-09-06F
18	8.2.1	See minutes 2002-09-06A but element retained for wrapping.
22, 23	4.4.4	See minutes 2002-09-06F
24, 25	4.5.2	See minutes 2002-09-06J
26, 27, 28	4.6.2	See minutes 2002-09-07
29	???	See minutes 2002-09-07

13 Author Information

Steven Tuecke
 Distributed Systems Laboratory
 Mathematics and Computer Science Division

Argonne National Laboratory
Argonne, IL 60439
Phone: 630-252-8711
Email: tuecke@mcs.anl.gov

Karl Czajkowski
University of Southern California, Information Sciences Institute
Email: karlcz@isi.edu

Ian Foster
Argonne National Laboratory & University of Chicago
Email: foster@mcs.anl.gov

Jeffrey Frey
IBM
Poughkeepsie, NY 12601
Email: jafrey@us.ibm.com

Steve Graham
IBM
4400 Silicon Drive
Research Triangle Park, NC, 27713
Email: sggraham@us.ibm.com

Carl Kesselman
University of Southern California, Information Sciences Institute
Email: carl@isi.edu

Peter Vanderbilt
NASA Ames Research Center
Email: pv@nas.nasa.gov

14 Acknowledgements

We are grateful to numerous colleagues for discussions on the topics covered in this document, in particular (in alphabetical order, with apologies to anybody we've missed) Malcolm Atkinson, Tim Banks, Ed Boden, Brian Carpenter, Francisco Curbera, Dennis Gannon, Andrew Grimshaw, Marty Humphrey, Keith Jackson, Bill Johnston, Kate Keahey, Gregor von Laszewski, Lee Liming, Miron Livny, Tom Maguire, Norman Paton, Jean-Pierre Prost, John Rofrano, Thomas Sandholm, David Snelling, Ellen Stokes, Scott Sylvester, Sanjiva Weerawarana, Von Welch and Mike Williams.

This work was supported in part by IBM and by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38 and DE-AC03-76SF0098; by the National Science Foundation; and by the NASA Information Power Grid project.

15 Glossary

16 References

16.1 Normative References

[RFC 1738]

Uniform Resource Locators (URL), T. Berners-Lee, L. Masinter, M. McCahill, Editors. Internet Engineering Task Force, RFC 1738, December 1994. Available at <http://www.ietf.org/rfc/rfc1738.txt>.

[RFC 2119]

Key words for use in RFCs to Indicate Requirement Levels, S. Bradner, Author. Internet Engineering Task Force, RFC 2119, March 1997. Available at <http://www.ietf.org/rfc/rfc2119.txt>.

[RFC 2246]

The TLS Protocol, Version 1.0, T. Dierks, C. Allen, Authors. Internet Engineering Task Force, RFC 2246, January 1999. Available at <http://www.ietf.org/rfc/rfc2246.txt>.

[RFC 2396]

Uniform Resource Identifiers (URI): Generic Syntax, T. Berners-Lee, R. Fielding, L. Masinter, Authors. Internet Engineering Task Force, RFC 2396, August 1998. Available at <http://www.ietf.org/rfc/rfc2396.txt>.

[RFC 2616]

Hypertext Transfer Protocol – HTTP/1.1, R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, Authors. Internet Engineering Task Force, RFC 2616, June 1999. Available at <http://www.ietf.org/rfc/rfc2616.txt>.

[RFC 2818]

HTTP Over TLS, E. Rescorla, Author. Internet Engineering Task Force, RFC 2818, May 2000. Available at <http://www.ietf.org/rfc/rfc2818.txt>.

[RFC 3023]

XML Media Types, M. Murata, S. St.Laurent, D. Kohn, Authors. Internet Engineering Task Force, RFC 3023, January 2001. Available at <http://www.ietf.org/rfc/rfc3023.txt>.

[Timestamp]

A Timestamp for Distributed Computing (Draft), D. Gunter, B. Tierney, S. Tuecke, Authors. Global Grid Forum OGSi Working Group. Available at <http://www.gridforum.org/ogsi-wg>.

[WSDL 1.2]

Web Services Description Language 1.2 (Working Draft). World Wide Web Consortium, July 2002. Available at <http://www.w3.org/TR/2002/WD-wsdl12-20020709>.

16.2 Informative References

[Globus Overview]

Globus: A Toolkit-Based Grid Architecture, I. Foster, C. Kesselman, Authors. In [Grid Book], 259-278.

[Grid Anatomy]

The Anatomy of the Grid: Enabling Scalable Virtual Organizations, I. Foster, C. Kesselman, S. Tuecke, Authors. International Journal of High Performance Computing Applications, 15 (3). 200-222. 2001. Available at <http://www.globus.org/research/papers/anatomy.pdf>.

[Grid Book]

The Grid: Blueprint for a New Computing Infrastructure, I. Foster, C. Kesselman, Editors. Morgan Kaufmann, 1999.

[Grid Physiology]

The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, I. Foster, C. Kesselman, J. Nick, S. Tuecke, Authors. Globus Project, 2002.
Available at <http://www.globus.org/research/papers/ogsa.pdf>.

[JAX-RPC]

Java™ API for XML-Based RPC (JAX-RPC). <http://java.sun.com/xml/jaxrpc/docs.html>

[Web Services Book]

Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI, s. Graham, S. Simeonov, T. Boubez, G. Daniels, D. Davis, Y. Nakamura, R. Neyama, Authors. Sams, 2001.

[WSIL Overview]

An Overview of the Web Services Inspection Language, P. Brittenham, Author. 2001.
Available at <http://www.ibm.com/developerworks/webservices/library/ws-wsilover>.

17 XML and WSDL Specifications

This Section will contain the full WSDL types, message, and portType for each of the operations described in this document. Watch this space.

Pending agreement from the OGSi-WG community on the directions and changes in this draft of the specification, the authors will produce formal WSDL and related XML definitions shortly after GGF5.